LATTICE®

FORTRAN Subroutines

PFALST(*BUFLOC,SIZE*) Set up allocation buffers.

PFSETP(*DLOC,ELOC,FLOC,GLOC*)
Set pointers for Data, XA, FCBs, PCB.

PFGET(*VARNO,PLOC*) Get variable into location.

PFPUT(*VARNO,PLOC*) Put variable from location.

PFGETI(*VARNO,PLOC*) Get 'I' variable into location.

PFPUTI(*VARNO,PLOC*) Put 'I' variable from location.

PFGETD(*FCBLOC,PLOC*) Get field data area into location.

PFPUTD(*FCBLOC,PLOC*) Put field data area from location.

PFGETF(*FNO,PLOC*) Get fcb into location.

PFPUTF(*FNO,PLOC*) Put fcb from location.

PFXSCN(*FCBLOC,XANO,SSNO,RESLOC,MAXN*)
Get extended attribute information.

PFPUTP(*TYPE,ENO,FLOC*) Insert new validation function pointer.

PFPUT/PFGET Variable numbers and sizes

1	PAFLX	2	12	PAMFCB	36	23 PAXTRA	17
2	PAFLXS	2	13			24 PAVAL	80
3	PASET	256	14	PARDHI	2	25 PAVLN	20
4	PANFL	2	15	PAECB	48	26 PVINCH	1
5	PAMULT	2	16	PAOLDH	2	27 PVDINCH	1
6	PAPSLO	2	17	PASSEC	2	28 PVNINCH	1
7	PAPSHI	2	18	PASCRO	2	29 PVKARG	2
8	PAOVTH	2	19	PABRD1	9	30 PVLARG	2
9	PAMLN	2	20	PABRD2	9	31 PAFSVN	2
10	PAMEN	2	21	PAKTS	2	32 PAXASN	2
11	PAMHD	max 80	22	PVWSTR	127		

PFPUTI/PFGETI Variable numbers and sizes

1	PAITRM	11	6	PAIEND	1	11 PAIIGP	4
2	PAIMAT	16	7	PAIBEL	2	12 PAIIGN	32
3	PAICAT	16	8	PAICFM	4	13 PAIKEY	65
4	PAIATT	16	9	PAIDL2	2		
5	PAIBOR	1	10	PAIPFX	4		

BASIC Calls

CALL **PSETUP**(*B%*) Set up Interpreter version (B%=1).

CALL **PSETUP**(*PSEG,POFF*) Set up Compiler version.

CALL **PANLOW**(*FUNC%,A1%,A2%,RET%*) Low-level functions.

CALL **PAUSEP**(*MLx%,ICx%,OCx%,IDx%(1),ODx%(1),*
IBx%(1),OBx%(1),XAx%(1)) Use panel.

CALL **PANELF**(*FUNC%,FLDN%,RET%*) Field-level functions.

CALL **PANELS**(*FUNC%,RET%*) Screen-level functions.

CALL **PAMSGE**(*MSG$*) Display message.

CALL **PAANSW**(*ANS$,SAT%,RET%*) Get Answer.

CALL **PASCNL**(*LINE%,SRCNL$*) Get screen line.

CALL **PASERV**(*FN%,ARG1%,ARG2%,ARG3$*) Service functions.

BASIC Low-Level Function Numbers

PAFUNC	1-10	PAGETF	15	PAHTYP	20
PACINP	11	PACURS	16	PAPAGE	21
PAROUT	12	PASBAD	17	PACHCL	22
PACSTA	13	PAHION	18		
PACOUT	14	PAHIOF	19		

BASIC Service Functions

1	Set PARDHI from ARG1%
2	Set PAMLN from ARG1%
3	Set PAHP150 from ARG1%
4	Get PAECB address to ARG1%, PAVDU address to ARG2%
5	Set/get PAXTRA from/into ARG3$
6	Set/get PAVAL from/into ARG3$
7	Set/get PAVLN from/into ARG3$
8	Set/get PAHPBUF from/into ARG3$
10	Set/get PAFLX from/into ARG2%
11	Set PAPSLO from ARG1%, PAPSHI from ARG2%
12	Set PAOVTH from/into ARG1%
13	Call PASCSE with ARG1%
14	Call PAMENU for field ARG1%, highlight ARG2%

PANEL

Reference Card

Roundhill Computer Systems Limited
POB 14 Marlborough, Wiltshire SN8 1LG England. Tel: (0672) 54675

Low-level Functions

Low-level functions are used for accessing the screen and keyboard directly, bypassing the PANEL field functions.

int **PACSTA***(void)* Obtain keyboard status, true if character available.

int **PACINP***(void)* Get a single keyed character without echo.

uint **PAGETF***(void)* Get a data character or interpreted key function from the keyboard.

void **PAROUT***(char)* Write one raw data character to the screen.

void **PAEOUT***(char)* Write one control or escape character to the screen.

void **PACOUT***(char)* Write one data character to the screen with highlighting.

void **PASOUT***(char *)* Write a null-terminated string using PAROUT

void **PAFUNC***(int)* Perform basic screen function: INIT(1), TERM(2), CLS(3), EEOL(4), BSC(5), BELL(6).

void **PACURS***(int,int)* Position cursor for reading from (column,row).

void **PASBAD***(int,int)* Position cursor for writing to (column,row).

int **PAHTYP***(int)* Set highlighting type and return highlighing method.

void **PAHION***(int)* Send raw highlight on sequence using modifier, if any.

void **PAHIOF***(void)* Send raw highlight off sequence.

void **PACHCL***(int,int)* Set attributes for next clear screen operation, border (IBM PC colour).

void **PAPAGE***(int,int)* Set active display page, working page (IBM PC Colour).

void **PALIFT***(int,int,int,int,char *)* Lift and save screen block from (x,y,l,h).

void **PARDSP***(int,int,int,int,char *)* Redisplay saved screen block to (x,y,l,h).

Field Functions

Field functions perform the requested function on a field identified by a passed field contol block.

int **PANELF***(int, panfcb *)* Perform field function (see list of codes below for functions).

int **PAMENU***(panfcb *,int,int,int)* Process multiple-line field as a vertical menu, returning selected line number.

Screen Functions

Screen functions operate on multiple fields, using the pointers to field control blocks in the PAFPT array to identify the fields to be used.

int **PAUSEP***(panpcb *,int)* Load PAFPT array with pointers to the fields in the specified panel.

int **PANELS***(int)* Perform screen function (see list of codes below for functions).

char ***PASCNL***(int)* Return text of screen line.

PANELF Function Codes

```
#define fdispu  1  /* display field */
#define fdisp   2  /* display field with highlighting */
#define fdispa  3  /* display field with HL & alternate border */
#define fbordd  4  /* display default border on displayed field */
#define fborda  5  /* display alternate border on displayed field */
#define funsh   6  /* redisplay saved screen from under overlay */
#define flose   7  /* forget and free saved screen under overlay */
#define freadu 11  /* read field */
#define freadh 12  /* read field with highlighting */
#define freads 13  /* read field from special start point */
#define fhigh1 21  /* set highlighting on for field */
#define fhigh2 22  /* complete highlighting for field */
#define fhighc 23  /* change highlighting of displayed field */
#define ferase 24  /* erase field from screen */
#define fsetm  30  /* set up mask for field */
#define ffilm  31  /* insert fill chars under mask control */
#define ffill  32  /* set field to fill characters */
#define funfil 33  /* set fill characters to blank */
#define fpadf  34  /* pad field data on right with fill char */
#define flnop  35  /* perform 'line operation' on field lines */
#define ffild  36  /* set entire data area to fill chars */
```

PANELS Function Codes

```
#define pdisp 101 /* display panel output fields */
#define pshow 102 /* display panel input fields */
#define pzero 103 /* clear input fields to fill character */
#define ppadf 104 /* pad input fields with right fill */
#define prjnf 105 /* perform 'line operation' on input flds */
#define pdelf 106 /* delete fill chars from input fields */
#define pread 107 /* read all input fields for panel */
```

INPUT Key Function Numbers

1	CR	9	INS
2	ESC	10	DEL
3	BS	11	LINS
4	LEFT	12	LDEL
5	RT	13	TAB
6	UP	14	QRY
7	DOWN	15	HOME
8	ERAS	16-	PF1-

Setup Functions

Setup functions are performed once at the beginning (or for PAOSTERM, the end) of a program to initialise the operation of the other PANEL functions.

void **PAOSINIT***(void)* Initialise the operating system (not required for MS-DOS).

void **PAOSTERM***(void)* Remove any operating system initialisation (not required for MS-DOS).

int **PAINIT***(char *)* Initialise PANEL functions.

int **PASCSE***(int)* Enable 'screen save' for field overlays.

Message Functions

Message functions operate on the designated message line for the panel.

void **PAMSGE***(char *)* Display message.

void **PAANSW***(char *,uint)* Retrieve answer into tail of message, using specified entry attributes.

Validation Functions

Validation functions are called by PANEL functions and should not be called directly by an application. The source is provided and may be modified as required to replace the library functions.

int **PAKVAL***(uint *)* Exit taken after every interpreted key depression.

int **PACVAL***(uchar *,uchar *,uchar *)* Exit taken after every data character during data entry.

int **PALVAL***(int)* Exit taken after every field line during data entry.

int **PAFVAL***(int)* Exit taken after every field during data entry.

int **PAXVAL***(int)* Exit taken before and after every field processed by the screen-level function PANELS.

Dynamic Load Functions

Dynamic load functions operate with screen panels loaded from disk at run-time.

int **PALOAD***(panpcb *,int,int)* Dynamically load requested panel, using full/abbreviated fcbs, with/without data space allocation.

int **PAFREE***(panpcb *)* Release space used by dynamically loaded panel details.

int **PAMLOP***(char *, char *)* Open file and index containing multiple panel specifications.

int **PALOOK***(panpcb *)* Look for specified panel in multiple file index, and prepare for next dynamic load to find it.

int **PAMLCL***(void)* Close file containing multiple panel specifications.

PANEL

(Version 6.20)

Roundhill Computer Systems Limited

PANEL(R)

Version 6.20

(April 1986)

Roundhill Computer Systems Limited

PANEL

PANEL is a product of:

Roundhill Computer Systems Limited
PO Box 14, Marlborough,
Wiltshire SN8 1LG, England

Telephone:	UK: National:	0672 54675
	International:	+44 672 54675
	USA:	(201) 569 2265
Telex:	UK: 444453 AWARE G	

Documentation

This is the PANEL documentation for version 6.20. It consists of an Introduction, a Reference Manual, and additional sections for specific languages and environments.

Disclaimer

Every effort has been made to ensure that this document accurately describes the PANEL system: however Roundhill Computer Systems Limited cannot accept any responsibility for the consequences of any discrepancies between this document and the system as implemented.

Roundhill Computer Systems Limited also reserves the right to make amendments to the contents of this document from time to time, to reflect changes made to the specification of the PANEL system or for any other reason.

Trade Marks

PANEL is a trade mark of Roundhill Computer Systems Limited. MS-DOS, MS and XENIX are trademarks of Microsoft Inc. CP/M and CP/M-86 are registered trademarks of Digital Research Inc. UNIX is a trademark of AT&T Bell Laboratories. IBM is a trademark of IBM Corporation. Lattice is a trademark of Lattice Inc. C86 is a trademark of Computer Innovations Inc.

Copyright 1985, 1986 Roundhill Computer Systems Limited

This manual is copyright. All rights reserved. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or stored on electronic medium or in machine readable form without consent in writing from Roundhill Computer Systems Limited.

Sections

I Introduction

II Reference Manual

III PANEL for MS-FORTRAN

IV PANEL for MS-BASIC

V PANEL for Amiga

VI Index

Table of Contents

I. Introduction 1
Welcome 2
On-Line Support 3
Overview 4
Installation 7
Getting Started 8
Using PANEL on IBM-PCs and Compatibles 9
Exploring PANEL 14
Using the Subroutine Library 20
Validation Functions 24
Using Levels and Overlay Fields 27
The Development Cycle 33
The PANEL User Interface 35
Questions and Answers 40
Screen Design Guidelines 47
II. Reference Manual 49
Introduction 50
Attributes 51
Borders 53
Environment 55
Extended Attributes 56
The **keyboard** Program 59
Numeric Validation 60
PAANSW 63
PACHCL 64
PACINP 65
PACOUT 66
PACRF 67
PACSTA 68

PACURS 69
PACVAL 70
PAECB 72
PAFLX 74
PAFPT 76
PAFREE 77
PAFUNC 78
PAFVAL 79
PAGETF 80
PAHTYP 81
PAINIT 83
PAKVAL 85
PALOAD 87
PALVAL 89
PAMENU 91
PAMSGE 93
PAMULT 95
The **panel** program 97
PANELF 102
PANELS 109
The **paneltc** Program 111
PANEL.VDU 112
panfcb 131
The **pangenc** Program 134
panpcb 138
The **pantest** Program 139
PAOSINIT 140
PAPAGE 141
PASBAD 142

PASCNL 143
PASCSE 144
PAROUT 146
PAUSEP 147
AXVAL 148
The **pd** Program 149
The **pnlindx** Program 153
PNL files 154
pnl*.h 155
The **tailor** Program 156
TAILOR.DAT 162
Validation 163
The **vdugenc** Program 167
Vertical Scrolling 168
III. PANEL for MS-FORTRAN 171
sing PANEL with MS-FORTRAN 172
IV. PANEL for MS-BASIC 191
Using PANEL with MS-BASIC 192
V. PANEL for Amiga 207
Using PANEL on the Amiga 208
VI. Index 219
Index 220

Section I
Introduction

Welcome

Welcome to the **PANEL** system. We hope that you will find it valuable, helpful, and fun to use. **PANEL** is designed to help you in the following key areas:

- **PANEL** enables you to design screen-based applications quickly and effectively, by letting you adjust field positions and sizes interactively, and providing flexible display and validation features.

- **PANEL** makes your programs easy to use by allowing extensive editing of operator input, and by helping you to achieve consistency in your screen layouts.

- the **PANEL** library includes functions which provide your programs with sophisticated display features such as pop-up menus and overlaid help boxes.

- screen designs and **PANEL** processing can be ported easily to all supported operating system and hardware environments.

- utility programs in the **PANEL** system allow you quickly to set up prototypes of your screen entry programs, to demonstrate to a client or user how the application will look.

The remainder of this section gives you an introduction to **PANEL** and how it is best used. The reference section lists commands, functions and other topics with full details of their operation.

This documentation is principally directed towards the 'C' programming language interfaces of the **PANEL** system. Separate sections describe the other languages supported by **PANEL**.

On-Line Support

By arrangement with Lattice, Inc., PANEL support is available through McGraw-Hill's Byte Information Exchange (BIX) Network.

f you already use BIX, *join* the *lattice* conference. The 'topic' used for PANEL support queries and information is currently *utilities* (if this changes, a message will be posted in *utilities* to this effect).

To subscribe to BIX in the USA, call Tymnet on (800) 336 0149 and ask for your local access point. Then dial into Tymnet and respond **a**, then **byteneti** at the login prompt, **mgh** at the password prompt, **bix** at the BIX login prompt and **new** at the BIX name prompt. Full details of registration procedures and fees will be given (and see also BYTE magazine). Registration and usage fees are billed to your Master Card or VISA account. After registration, you can *join lattice* as described above.

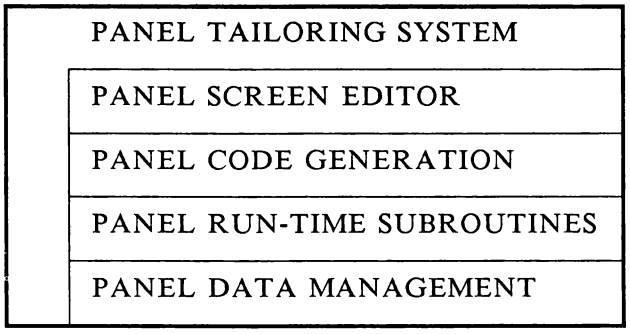
Outside the US, follow local procedures for access to Tymnet, and then proceed as described above. In the United Kingdom you can set up a British Telecom Packet Switchstream (PSS) account (call inkline 0800 282852 for information). Once on PSS, the Tymnet ddress is entered as **A93106**.

Overview

This section aims to answer your first questions about PANEL and its capabilities. Because PANEL is a product which is designed to help programmers to write programs quickly and with a high-quality user interface it follows that most users of this manual will have at least some programming experience. For this reason you will not find in it sections teaching you about operating system prompts and how to switch your computer on or off. Instead we have attempted to strike a balance between explanatory narrative, examples, hints about the use of the system, and detailed technical information which you will need as you use more of PANEL's features.

PANEL's main components.

The PANEL System consists of five components as follows:



The **TAILORING** component, as well as including the 'install' program for the PANEL system itself, also allows your own application to be run with independence from both the screen and the keyboard in use in the target system.

The screen **EDITOR**, named **panel**, is used to set out and modify screen panel layouts to suit your application. Both input and display fields can be defined, on multiple 'levels', and the program permits a wide variety of field attributes to be specified.

The **CODE GENERATION** facilities of the PANEL system generate both data definition statements to describe the screen fields used by

your application, and also control blocks initialised with the position, size and attributes of each field. The idea is that screen ata should be shared between PANEL and your application, esulting in efficient and easy-to-maintain programs.

PANEL **RUN-TIME SUBROUTINES** provide the means of bringing a high level of user-friendliness to your application, and allow the end-user many useful editing facilities during data entry. The source code for entry validation is included, so that data entry can be totally under control of the system designer.

The **DATA MANAGEMENT** component, while not designed as a full database system, can be used either for record-keeping and file indexing purposes, or can also be configured as a data-entry front-end to another application. The source is also provided, but requires a proprietary file access package to provide indexing functions.

Functions of PANEL programs

his section describes the function of each program in outline. Fuller details are of course available in the appropriate section of the reference manual.

The **tailor** program is used to select the operating environment for both PANEL system programs and those developed incorporating the subroutine library. It does this by creating a file named **PANEL.VDU** which contains specifications of the screen and keyboard in use. **tailor** is used both to select specifications from its own master data file and to edit the details. Programs can also obtain the screen and keyboard data from the **TAILOR.DAT** master file, without the need to access an individual **PANEL.VDU** file for each user.

In operating system environments which support a 'termcap' terminal capability database, the **paneltc** program provides a quick ethod of creating a **PANEL.VDU** file without the need to look up escape sequences and keyboard codes. The file created can then be enhanced with **tailor** to add additional keyfunctions or highlighting features.

Overview

Because your programs developed using PANEL may make use of a number of function keys, it is useful for an operator to be able to check the current function key assignments. The **keyboard** progra reads the **PANEL.VDU** information and shows the keytop legends recorded there.

The **vdugenc** program generates a module describing the current screen and keyboard. This module can be compiled and then linked with your application programs to avoid the need to keep a copy of **PANEL.VDU** on the disk.

panel is the main program used for creating and modifying screen panel descriptions.

After a screen panel layout has been created, the **pantest** program can be used to test and document it. The program loads the panel dynamically and allows sample data to be entered. It also permits 'snapshots' of the screen to be written to printer or disk. This program is also provided in source form as an example.

The data management function of the PANEL system is handled b the **pd** program. This program can read a screen panel description and create a data entry or record-keeping system based on the screen specifications. For some versions of PANEL we are unable to include an executable version of **pd** because of the terms of our licence of the file handler which it uses. The source of the program is, however, included with all versions.

The **pangenc** program generates program code to describe a PANEL generated screen layout, which is included or compiled and linked with your application code.

Installation

If you have a system with only floppy disks, first make a copy of the distribution diskettes and place the originals in a safe place. Follow the principles outlined below to copy the PANEL system files onto your work disks in the most convenient manner.

Most UNIX and XENIX versions of PANEL are distributed in *tar* or *cpio* format: if this is not the case the distribution medium will so indicate. The first step is to unload the contents of the distribution medium into a temporary directory and change the file ownership and permissions to match installation standards.

PANEL header files should be placed in the standard place used for other include files. The library or libraries should also be placed with other library files. The command files should be moved to a directory which can be accessed through the standard PATH. We recommend moving the remaining files into a directory named 'panel', and setting the environment variables MSG and TAILOR to enable them to be found there. The MSG environment value should be the panel directory name, terminated by the path separator haracter to indicate that it is a directory: the TAILOR environment value is the full path and file name of the **TAILOR.DAT** file.

For single-user systems, the **TAILOR.DAT** file will not normally be required after the tailoring process has been completed. Instead the **PANEL.VDU** file (if required, see next section) should be placed in a convenient directory and the VDU environment variable set to its full path and filename.

If your compiler supports options which require separate PANEL libraries (eg 8086 memory models) you should identify the library required and the corresponding **pnli.h** file and build your 'make' files so that the correct files are selected for your application.

Important Note. If your operating system includes a *'ranlib'* rogram, run it on each PANEL library file (**panelxx.a**) after the file has reached its final resting place.

Getting Started

After the PANEL system has been installed, it is normally necessary to tailor it for the screen and keyboard in use. For single-user systems such as MS-DOS, PC-DOS, and CP/M-86 it will usually onl be necessary to run **tailor** and 'pick' the appropriate set of tailoring parameters from **TAILOR.DAT** using the provided menu. If some of the terminal and system names seem a little terse, you can type an entry number followed by a question mark to display (at the top of the screen after scrolling) any further explanation which is recorded in the principal comment field for the entry.

For details of installation options for the IBM Personal Computer and compatible systems, see the next following subsection. Other environments which do not need to use the PANEL tailoring facilities are AmigaDOS and QNX.

The UNIX and XENIX versions of PANEL may also include an appropriate terminal entry in the **tailor** menu, but if not, run **paneltc** to create **PANEL.VDU** from the termcap database. The resulting file will include basic highlighting but you may wish to use **tailor** to add additional highlighting or colours not included in termcap. In addition you should use **tailor** to redefine key functions from the universal standard set provided by **paneltc**. This is simply a matter of pressing the selected keys to define them to the **tailor** interactive keyboard definition function.

If all else fails you will have to run **tailor** to define all of your screen and keyboard functions. Read the sections on **tailor** and **PANEL.VDU** in the PANEL reference manual and follow **tailor**'s on screen instructions.

Finally, run **keyboard** to display the defined key functions. You may find it useful to redirect the **keyboard** output to the printer to create a reference list.

Using PANEL on IBM-PCs and Compatibles

Introduction

ince version 6.01, PANEL has provided three ways of running programs on an IBM Personal Computer or compatible system. In order of increasing display speed, these are as follows:

- Using the ANSI terminal driver.
- Using the ROM BIOS functions.
- Direct I/O to the screen regen. buffer.

This section provides additional information about using and customising each of these three options. Please note that in the rest of the section we shall use *IBMPC* to refer to an IBM Personal Computer or a fully compatible system. Throughout this section we assume operation under PC-DOS or MS-DOS: other operating systems for the IBM PC are usually covered in a 'readme' file on the distribution medium.

Selecting the display method

The factors affecting the selection of which method to use for screen display are performance, portability, and hardware and software compatibility. To make your selection, we recommend reviewing the following list:

- To use the *direct* method, you must link your application with an optional 'I' library as described below. Such an application will then only run on hardware which has a memory mapped display in the same position as an *IBMPC*.
- The *direct* method is substantially faster than any other.
- When you link your application without the 'I' library you retain the maximum flexibility. You can use the PANEL tailoring system to select either the *ANSI* or the *ROM BIOS* method without changing the application .exe file. The same application will also run on non-IBM-compatible systems and terminals.

- The *ANSI* method is often much slower than the others, but it may enable you to retain compatibility with certain non-standard operating environments. Note that the presence of ANSI.SYS in your CONFIG.SYS file does not require you to use the *ANSI* method if others are available.

- The *direct* method produces programs which are 'well-behaved' under Microsoft Windows or IBM Topview, and which will run efficiently in a window which can be moved and sized.

- You may be able to modify the low-level source routines provided for the *direct* method so that your application runs in memory-mapped mode on non-IBM-compatible hardware.

Customising the ANSI parameters

Select the 'IBM PC ANS' parameters using **tailor**.

The *IBMPC* ANSI screen attributes are controlled by escape sequences in the same way as a terminal. To change the colours, you can select other SGR (set graphics rendition) sequences either for the clear screen or the attribute on and off sequences using **tailor**. The PANEL highlight method code should be set to 4, except for the highlight type which is to be used for messages, which is set to 132.

You can use the highlight modifier to alter colours by inserting a special 'M' code in the escape sequence before the second byte of the colour number. The modifier will be or'ed with the second byte of the number, which should normally be set as '0'.

Customising the ROM BIOS parameters

Select the 'IBMPC C80' or 'IBMPC MONO' parameters using **tailor**. The five letters 'IBMPC' at the start of the name indicate that the ROM BIOS calls are to be used.

The *ROM BIOS* screen attributes are controlled by the INIT sequence and the attribute ON sequences. The three bytes of the NIT sequence indicate the screen mode, the base physical attribute for screen clear operations, and the border colour attribute. We use the same attribute values as defined in the IBM Technical Reference Manual, and a copy of the colour values also appears under PANEL.VDU in the reference section of this manual.

You can select a physical attribute for each of the 16 PANEL highlight types and enter it as the appropriate 'attribute on' sequence using **tailor**. In each case the 'off' sequence is not used, and the method code should be 4 except for the highlight type to be used for messages, which should be 132.

We recommend entering the physical attribute in hexadecimal (by following it with an 'H' in **tailor**), so that it is easy to distinguish the background colour in the first nibble from the foreground colour in the second.

he highlight modifier for the *ROM BIOS* method is actually or'ed with the physical attribute. You can either specify a zero attribute and control the colour entirely with the modifier, or you can select foreground or background with the attribute and the other half with the modifier.

When the *ROM BIOS* method is in use the **PAPAGE** and **PACHCL** functions select display pages and border and clear screen attributes as described in the reference section.

Note that some other **tailor** parameters starting 'IBMPC' are used for systems which are nearly compatible at a ROM BIOS level. The 'CMPQ' parameters are used for early Compaq systems, but later ones seem to work fine with the normal 'IBMPC' parameters.

Customising the Direct I/O Parameters

The 'direct' method makes no use of the PANEL.VDU file. All the customising of screen attributes is done through tables in the program. These tables can be modified by use of DOS environment specifiers as well as by program. The tables appear in the low-level

Using PANEL on IBM-PCs and Compatibles

I/O module **PNL1*nn*2C.C**, and can either be modified dynamically by your program or by recompiling this module. If you refer to *PAI....* variables in your main program, then it must obviously be linked with the 'I' library. If you recompile the module, remember to define the symbol **PAMMAP**.

The screen display attributes are selected from either a 'monochrome' or a 'colour' set (**PAIMAT** or **PAICAT**) by testing the hardware screen mode during **PAINIT** processing. The appropriate parameters are placed in the **PAIATT** table. The table has 16 bytes corresponding to the sixteen PANEL highlight types, and each byte contains an *IBMPC* physical attribute. The first byte is used for the default, or base, screen attribute, and the second is selected for messages. There is no 'method code'.

Environment specifiers are used as follows:

- The PNLMONO environment variable can be set to a non-null value (at least one blank) to force the selection of the 'monochrome' attribute set on systems with a mono graphics CGA-compatible display.

- The PAIATT environment variable can be set to a hexadecimal string which will replace the **PAIATT** array. As an example, the command *SET PAIATT=120f56* will select '12' as the first entry (green on blue), '0f' as the second (bright white on black) and '56' as the third (yellow on magenta). Remaining entries in the **PAIATT** array, if defined, will be unchanged.

- The PAIBOR environment variable can also be set with a hexadecimal (two-character) value. For example *SET PAIBOR=04* will select a red border.

Further information on customising the *direct* parameters appears in the low-level source module. Note that if you increase the size of the **PAIKEY** array to add additional function keys you must also maintain **PAIIGN** as half its size, and increase **PAKTS**. **PAKTS** is set to the total number of bytes in the table, less the 40 bytes of the 'standard' table.

Alternate Utility Programs

The PANEL distribution disks for PC-DOS and MS-DOS versions contain alternate versions of the **panel**, **pantest** and **testc** programs named **paneli**, **pantesti**, and **testci**. These programs are for use on *IBMPC* hardware only, and run much faster than the normal versions of the programs. Do not use these special programs on hardware which is not IBM compatible. The 'I' versions make no use of the PANEL.VDU file.

Linking Programs for Direct Memory-Mapped Operation

When applicable, additional libraries are provided with an 'I' suffix after the interface number. When you wish to link your program to use the *direct* screen method, name the 'I' library to your linker before, and in addition to the normal library. The memory-mapped I/O routines will then be used instead of the low-level routines in the normal library. The resulting programs will only run on *IBMPC* hardware.

Exploring PANEL

The PANEL user interface

We recommend that you start by running the **testc** program, of which the source file is supplied. This program displays and then reads the **demo** screen panel layout, and illustrates several features of the PANEL user interface. The program was created by linking the compiled testc source file with compiled code generated for the **demo** panel by **pangenc**. You may find it useful to refer to a printout of the source file while running the program.

Provided that it found the **PANEL.VDU** information, the program will start by displaying the entry screen. Before you enter any data, notice that the program has used separate calls to **PANELS** to display the output fixed fields and then the input fields. It displayed by default the fields at 'levels' 0 and 1; levels 2 to 4 of the **demo** panel are not used. After displaying a prompt message, the **PANELS** function is called again to read the whole screen.

The first two fields have attributes which convert input to upper case and which require 'carriage return' when the field is full. In addition. existing data will be 'wiped' if you start entry other than by forward spacing into the field. If you press the 'help' (usually "?") key during entry, a help message is displayed which disappears as soon as you restart entry.

When you move down (with <cr> or cursor down) to the address field, you will find that the 'help' message has become a 'help box'. The name and address is a multiple line field and you can use the cursor keys to move about in it. Try also the keys defined for insert and delete character, and (at the left-hand edge only) the insert and delete line keys. 'Home' takes you to first to the start of the line, then to the start of the field, and finally to the first field of the screen. The 'tab' key provides a 'word tab' function, then skips to the next line or field.

The name and address field has also been used as a demonstration of the PANEL 'vertical scroll' feature. Behind the five-line field on the screen is a ten-line data area. As you move the cursor out of the bottom or top of the field, any lines not displayed will appear. When the top line of the field has been scrolled out of the display

window, and the cursor is in the top left display position, the 'home' key will redisplay the field from the top.

The next five fields illustrate different types of entry which have been specified with the **panel** program. The telephone number is a 'mask' field, in which entries are only allowed in certain positions. The 'salesperson' field is a selection field which displays a list of values as the space bar is depressed. Try the help key on this field for additional instructions. The 'code' uses 'picture validation', again explained by its help message, and the next field makes use of another 'values list' to emit a string on typing its initial letter. Finally the numeric field is set up with 'currency', 'right-justify', and 'wipe' attributes.

The last multiple-line field is defined with a word-wrap attribute, which operates on initial entry only. PANEL does not include a facility to reformat a word-wrap field (but you could write one as a validation exit).

fter reading the whole panel, **testc** demonstrates how to read a single named field, using the **PANELF** call. Finally, it calls the **PAMENU** function for a multiple line field called VMENU. To select an item, move the highlighted bar with the up and down cursor keys, or key the initial letter of the item. After you have made a selection, the pop-up menu disappears, and **testc** then reports some of the items you entered, using ordinary **printf** calls on the generated data areas.

To complete this view of the PANEL user interface, enter

pangenc -f demo

to produce the **demo.c** and **demo.h** files used by **testc**, and make a printout of them to see how the fields are described. The structures used by these files are defined in **pnl3.h** and **pnl5.h**.

ynamic Loading of Panel Layouts

Another view of the PANEL interface is provided by the **pantest** program. This is not linked with generated code for a screen layout, but uses the PANEL dynamic load function **PALOAD** to

read one at execution time. The source of this program is also provided, and you may wish to list it out for reference. The program may be given an argument on the command line, or if not will ask for the panel name: enter **demo**.

As with **testc**, the **pantest** program displays and then reads a screen panel layout, but there are some differences. First, the program uses the option to set a highlighting type in **PARDHI** which will be used to highlight each field as the cursor moves to it. Secondly, only 'level 0' fields are initially displayed (by setting **PAPSHI** to zero) and so the box in the top right corner is not overlaid as it is in **testc**. Finally, the 'name and address' field in the **demo** panel is not set up to scroll vertically. Other than this the first stage of the program operates exactly as it does in **testc**.

After the screen entry has been completed, **pantest** presents further options. You can repeat the entry operation, with or without clearing the entry fields first. You can print the screen layout, or write it to a disk file (for your own panel layouts, this feature can be used to help with the creation of documentation for your users) And you can key a level number to display any fields defined by **panel** at that level.

The **demo** panel has additional fields at levels 1-4 and 9. You can display each of these in turn by keying its number. After displaying each level, **pantest** returns to entry mode at level 0 and rewrites the input fields: use escape or a function key to return to the main prompt to display another level. The field at level 4 is an entry field, so **pantest** requests input for it. In each case the fields are lifted from the screen after processing has been completed. The menu field on level 9 is treated as a normal field and **PAMENU** is not called for it.

The **pantest** program can be used to load several screens in one session, and these screens may be in separate files or collected into one file. These features are described in the reference section.

The PANEL Editor Program

The next step in our exploration of the PANEL system is to see how the screen layouts are designed and specified. The **panel** program is

the one to try next, with an argument of **demo** (or you can enter **demo** when the program requests a panel name).

The demonstration panel is again displayed, but this time for modification of the layout or attributes, and not for data entry. The cursor starts at the top left and can be moved with the arrow keys, the space bar, tab, carriage return or home. At almost any time while using **panel** you can press the help key (usually "?") for context sensitive help information. Start by seeing how the 'level' mechanism works: keying any of 1, 2, 3, 4 or 9 will display the defined fields at that level. The current level number, if non-zero, is displayed alongside the panel name. The program displays the fields at all levels 'visited', but the higher levels can be removed by returning to level zero. You can overlay fields as required by selecting levels in a particular sequence. Return to level 0 to experiment with some of the basic **panel** functions.

Move the cursor to the name and address input field and key 'S' to select it. The submenu gives you a number of additional choices.
he 'A' option abandons your selection and removes the highlighting. 'B' and 'E' reshape the field after you have moved the cursor to a new start or end point. 'C' and 'M' copy or move the field to the cursor position, and 'D' deletes it after a warning prompt.

The second type of field selection is made by keying a 'D' when the cursor is in a field. This 'define' option allows the field to be named, initialised, and given various attributes. Naming a field is essential if you plan to use it by generating code, since only fields with explicit names appear in the include file. A non-blank fill character is useful if an entry field is not highlighted.
Initialisation is primarily used for prompting fields but can also be used for entry fields. The 'A' option on the submenu steps through entry attributes, validation parameters, and highlighting. Finally the 'X' opens an area of approximately 1kb in which you can enter extended attribute specifications. (Exit from extended attribute specification by using the 'escape' key). We recommend that you try all of these options to examine what is defined for some of the fields in the **demo** panel and then read the reference section before defining your own fields.

Exploring PANEL

To set up a new field at the current level, 'B' is keyed at the main edit prompt, with the cursor at the required position, followed by an 'E' after moving the cursor to the end of the field. You can check the exact size of a new field by keying 'I' with the cursor in the field. ('I' keyed outside a field gives information about the whole panel). When you have set up attributes which are to be the same for a number of fields, you will find it quicker to copy the first one and reshape it as required than to set up the same attributes again for each.

There are many other **panel** features which you can try at this time. Whole blocks of fields can be moved by using the character or line insert and delete keys with the cursor position defining the affected block below and to the right of it. Only fields at the current level participate. Fields can be moved or copied from one level to another by changing levels after you have selected the field. A border can be defined by giving a field a 'B' attribute. (The border is not a part of the field when you come to place the cursor to select it).

If you wish to try writing out the panel layout, we suggest you change its name if you have modified it. After 'escape' from edit mode, select the 'W' option and enter a new name. (If you want to change it, for example to 'DEMOXX', the tab key will take you to the end of the word 'DEMO'). After entering any comment, you will be asked how to sort the panel fields. The screen display and read functions treat fields as a linear array, which either operates basically from left to right, or from top to bottom. Despite appearances, the **demo** panel is actually sorted horizontally, so that the 'notes' field comes at the end. Fields starting in the same column are sorted from top to bottom as a subsidiary sort 'key' in the 'horizontal' sort.

PANEL Data Management

The last program to try out is the **pd** data management program. Run it on the **demo** panel to create a data file **demo.dat** and index file **demo.idx**. The index fields are the 'contact' and 'company' fields (defined with a 'K' attribute).

First you need to add some data, which is done simply by typing in each screenfull and selecting 'F' to file the data. After adding a few records, escape from the addition process and key 'R' to retrieve it.

Each retrieval starts by entering one key value or partial key value terminated with a backwards oblique. You can then scan forwards or backwards through the file by keying 'F' or 'B'. Records can be amended, deleted, or printed as required.

Using the Subroutine Library

Naming Conventions

All PANEL library module names start with 'pnl'. If the following characters are alphabetic, then the module is either a validation module or one of the utility routines such as that provided for processing extended attribute substrings (**paxascc**). The final character of the module name always indicates the source language as follows:

c	'C' language
a	Assembler
f	FORTRAN
p	Pascal

If the character following the 'pnl' is a number then the module is part of the main subroutine library. The next two characters indicate the 'interface' number for modules which are used in only one environment. The characters '00' here indicate a 'common' module.

Common Library Modules

PANEL library modules are numbered as follows:

pnl1xx1	Low-level PANEL functions
pnl1xx2	Low-level I/O functions
pnl1xx3	PANEL.VDU information
pnl1xx4	Alternate low-level PANEL functions (eg Memory-mapped I/O)
pnl1xx5	Alternate low-level I/O functions
pnl1xx6	Miscellaneous low-level functions
pnl2xx1	Field operations
pnl3xx1	Interface between PANEL and application
pnl5xx1	Screen-level operations
pnl5xx2	Low-level I/O for dynamic load

pnl5xx3	Dynamic load
pnl5xx4	Multiple dynamic load
pnl5xx5	Screen Dump
pnl5xx6	Message/Answer
pnl6xxx	Miscellaneous

Function Overview

Only two function calls are used to carry out the majority of PANEL operations. The **PANELS** call operates on a whole screen, or section of a screen, and the **PANELF** call operates on a single field. Your application can mix these calls as required. These calls are used to display and 'read' (from the operator) screen fields, and to perform a number of utility functions on the screen data areas. Each operates through a 'field control block' structure or structures; in the case of the field operations it is directly passed as an argument, and for full-screen operations the field control blocks are accessed though a pointer table called **PAFPT**. This pointer table is ormally set up with the **PAUSEP** (for 'use panel') function.

Screen details can either be generated (with **pangenc**), compiled, and linked with your program, or can be loaded at run-time with the **PALOAD** function. In either case, you must still call **PAUSEP** before making calls to **PANELS**.

Before making any other PANEL calls there are a series of initialisation steps you must perform. The **PAOSINIT** (and later, **PAOSTERM**) functions should be used when operating system defaults need to be changed at the start and end of a program. Then **PAINIT** must be called to initialise the PANEL screen functions. Next, **PASCSE** will enable the screen save functions which implement PANEL pop-up help boxes, menus and overlaid fields. Lastly, the **PAFUNC(1)** call (and later, **PAFUNC(2)**) should be used to physically initialise the screen or terminal. Languages other than 'C' may require additional initialisation calls which are described in a later section.

PANEL low-level functions are used less frequently in most programs, apart from the **PAFUNC(3)** call to clear the screen.

The screen and field operations have a number of 'exits' which are taken under different circumstances to perform validation and other processing. These exit functions are described in a later subsection.

Other PANEL library functions are provided for specific purposes, such as processing 'messages' and answers (**PAMSGE** and **PAANSW**), pop-up menus (**PAMENU**), and multiple screens in one dynamic load file (**PAMLOP**, **PALOOK**, and **PAMLCL**).

A full list of the PANEL functions in each of the above categories will be found on the reference card. In addition, many examples of their use will be found in the supplied source files. The reference sections of this manual give the full syntax for each call.

Sharing data with PANEL subroutines

PANEL is designed so that, wherever possible, data for display or after entry is shared between your application and the PANEL routines. The principal means of achieving this is the pointer in each field control block to the data area for the field. This data area is the place where you place your data for display, and to which PANEL will return data entered by the user.

When **pangenc** has been used to generate code for the screen layout, there will be a named data area for each screen field to which you have assigned a name with the **panel** editor. When the screen is dynamically loaded, however, the field name is not directly available to access the data. There are three main ways to handle data in this case:

(a) Use the field number, and access the field data through the fcb pointer in **PAFPT** for the field number and the **datablock** pointer in the fcb.

(b) Generate code for the PANEL using **pangenc** with the '-e' switch and discard everything except the generated *enum* set. Include this to cause your compiler to translate the field name to a number which you can use as in (a).

(c) Load the panel with the **PALOAD** option for full field

control blocks, and scan through the field names to convert to a field number. If you write a function to do this, remember that the field names in the fcb are not null-terminated.

Linking your programs

When you actually come to link together programs you have created incorporating PANEL, it is not always obvious how to build the executable program. We list below typical linkage command lines which we have found work with the current versions of some of the compilers we support. In all cases, we have chosen the simplest command options available, on the assumption that if you can select the other options you will also be able to work out how to modify the command lines to match.

The examples shown create the testc program, assuming that the code generator **pangenc** has already been run to create *demo.c* and *demo.h* and that *testc* and *demo* have been compiled. Even if your ompiler does not appear in the list, you may still be able to use the examples to work out how to enter the command for it. For simplicity, we assume that all files are in the same directory, though this will seldom be the case.

Generic Unix	*cc -o testc testc.o demo.o panel99.a*
Zilog Zeus	*cc -i -o testc testc.o demo.o panel51.a*
Xenix 286	*cc -i -o testc testc.o demo.o -lx panela1.a*
Lattice C	*link cs+testc+demo,testc,,panel40+lcms+lcs*
CI C86	*link testc+demo,,,panel17+c86s2s*
DeSmet C	*bind testc demo panel35.s*
MW C86	*cc -f testc.o demo.o panel70.olb*
Wizard C	*link c0+testc+demo,,,panel73+clib*
Aztec C	*ln testc.o demo.o panel28.lib m.lib c.lib*

(The compiler names shown are generally trade marks of their authors).

Validation Functions

Introduction

There are five types of validation exits from the PANEL library. The first three are exits from field operations and the last two from screen operations.

PAKVAL is the first exit taken for each key depression. The key will have been decoded into the appropriate PANEL function or a data value before the exit, and the function can be transformed or rejected in various ways. If the key is a data value the next exit **PACVAL** is then taken, during which the appropriate data checking for the field can be carried out. Finally, the **PALVAL** exit is taken after each line of a field, and is normally used for number or date reformatting or validation. The field control block is available during each of these exits, and any PANEL function other than another read can be performed 'recursively'.

From the screen operations there are two exits. **PAFVAL** is taken after each field, and its normal use is to modify the route through the screen. **PAXVAL** is a general-purpose exit taken before and after each field. Details of the provided standard validation functions are given in the reference section.

From version 6.20, the PANEL validation functions have been reorganised into separate functions driven through a function pointer table. This makes it very much easier to remove unwanted functions or to add new ones. In earlier versions of PANEL it was necessary to keep special modified versions of our large validation source files.

Each provided standard validation is now implemented in a separate source file (some source files contain more than one type of exit function needed to implement a single validation type). The functions are selected through a master function pointer table in **PNLVPTBC.C** (C versions) or **PNLFVPTC.C** (MS-FORTRAN). This approach has three advantages:

- You can remove unwanted validations by deleting them from the table or by including a cut-down version of the table in your own program.

- You can add new validation functions very easily by writing a new module and adding a declaration and table entry in the pointer table module.

- A program can dynamically use a local validation function for special purposes at a particular stage of data entry, simply by temporarily loading a pointer to it into the appropriate table entry. An example of the syntax for this is given in the table module.

The comments in the validation pointer table source module list the file names which contain each of the validation functions. In general the modules have the same name as the validation function except that the module name starts *PNL* and the function name starts *PA*. The main modules for each of the five types of validation are **PNLCVALC**, **PNLKVALC**, **PNLLVALC**, **PNLFVALC**, and **PNLXVALC**. Each of these modules actually calls the appropriate numbered function through the function pointer table.

Support routines

A number of validation support routines are provided in separate modules. The functions are as before fully described in the comments, but they are listed here for reference:

PNLNUMVC Contains the numeric field line validation function.

PNLPSLVC This module processes the **PAPSLO/PAPSHI** selection for the **PANELS** function.

PNLRSTCC Reset the cursor at the end of a validation operation which may have moved it.

PNLTDSPC Test for a displayable character.

PNLXASCC Extended attribute scan for substrings.

In addition the **PAMENU** function is in **PNLCV09C** which provides validation functions for menu processing.

Three additional functions, **PABM**, **PABF**, and **PAIX** are provided

for low-level move, fill, and byte index operations respectively. These are not documented in the PANEL Reference Manual but the source is provided in the low-level module **PNL1*nn*2C.C**. They use functions normally found in compiler libraries but generally not with any consistent name or syntax.

Validation parameters

To simplify the function calls, the arguments to each of the validation functions are actually passed to the numbered validation function as static global variables. These variable names begin with *PV*.... and are described in the header file **PNL4.H** as well as in the reference manual.

Using Levels and Overlay Fields

Introduction

The availability of more than one 'level' of overlapping fields has already been mentioned; this subsection describes the design aims of this feature and gives examples of the uses of overlaid fields.

Note: 'overlay' in this sense has nothing to do with 'program overlays'.

Levels

'Levels' are numbers assigned to groups of fields on a screen panel which can be used to subdivide the panel. We use the term 'level' because individual fields can conveniently be thought of as 'on top of' or 'underneath' other fields.

There are two main reasons for defining fields on more than one level. First, you may wish to subdivide a single panel into sections processed at different times in your program. In many cases this could equally easily be done by defining several separate panels, but it is often convenient to group the fields on a single panel using levels. Second, you may wish to have fields which temporarily overlay other fields. Since the **panel** editor program does not allow overlapping fields on the same 'level', you must use separate level numbers to define such fields. You may like to think of these fields as 'transient' or 'pop-up' fields.

Individual fields can be processed by the **PANELF** field-level function without regard to their level number. When using the **PANELS** screen-level function, you must set variables **PAPSLO** and **PAPSHI** with the range of level numbers you wish to process. These two variables are integers defined in the PANEL library, and default to zero.

Field Overlays

The PANEL technique for overlaying fields should be considered separately from the use of levels, although the two are related concepts.

In overlay mode, PANEL field display routines will 'save' the

current screen contents at the position for the displayed field, prior to actually displaying the field. This allows the contents to be restored at a later time. It is important to note that it is your program's responsibility to deal with the saved screen data, either by redisplaying it or by releasing it.

The PANEL field display function uses the 'overlay mode' if three conditions have been met:

- The **PASCSE** routine has been successfully called to enable screen save processing.

- The level number of the field is higher than the value in **PAOVTH**, the overlay threshhold.

- There is space available to save the previous screen contents.

These conditions are discussed in the following sections.

Screen Save Enable

The **PASCSE** function enables 'screen save'. This feature maintains a copy of the screen contents in memory so that a request to save a section of the screen can determine what the current contents are. The argument to PASCSE determines whether the highlighting (and modifier) are saved with the screen data. A value of one saves only the data, two indicates data and highlight type, and three adds the highlight modifier.

On an IBM Personal Computer using either ROM BIOS calls or the 'I' library (see below) the physical attribute is saved instead of the highlight type and modifier. In this case the routine can also be called with a zero argument, which enables screen save but uses the actual screen memory to determine the screen contents, without allocating any extra storage areas.

Overlay Threshhold

The key to successful use of the PANEL overlay field feature is the **PAOVTH** variable. This is an integer variable defined in the

PANEL library which can be set by a user program. The default value of this variable is 1.

f no change is made to the default **PAOVTH** value, fields with a level number higher than 1 are displayed in overlay mode, and fields with a level number of 0 or 1 are displayed normally.

Note that changing **PAOVTH** has no effect on fields already displayed on the screen. Examples of the use of **PAOVTH** are given below.

Screen Save Pointers

When the 'current screen contents' have to be saved during a field display in overlay mode, memory is allocated dynamically as required. Obviously, if no memory is available the display fails and an error code is returned. In addition, we provide a fixed-size array of pointers for use with this memory allocation function, and if there are no unused pointers available the display will also fail.

The default size for the pointer table is 16, and this means that a maximum of 16 fields can concurrently be displayed in overlay mode. This will probably be enough for most applications which are using field overlays for transient display. The table size can be increased by changing a define of **NSAVES** in module **PNLXASCC.C** to the value required (up to 127). Note that in the Amiga versions of PANEL the pointer table is replaced by an array of structures incorporating the actual BitMap structure used for the save.

Your program must manage the screen save pointers using **PANELF** calls. Each *fdisp* function consumes one pointer when overlay mode is used and each successful *funsh* or *flose* function releases one. The *funsh* function redisplays the previous screen contents before releasing the memory and the pointer, and the *flose* merely forgets about the saved screen contents entirely. *Important Note*: If you merely clear the screen and begin something new in your program after displaying fields in overlay mode, you will soon run out of screen save pointers or memory.

A short-cut is available to unshow or 'lose' a number of fields. You

may call **PANELS** with an argument of *funsh* or *flose* to perform the field function on a range of fields. For the *funsh* function the fields are processed in reverse sequence, to attempt to ensure that the fields are removed in the correct sequence (see below). For a complex arrangement of overlays this technique may not be successful.

In calculating your requirements for screen save pointers, remember that a help box display uses (and frees) one pointer and that the **PAMENU** function uses (and frees) two pointers.

Using Level Numbers to Subdivide a Screen

The simplest use of level numbers is to break up a screen into sections, displayed at different stages of a program. Using separate panels selected with **PAUSEP** is another way of doing this. In this case, it is important to set **PAOVTH** to a higher level number than any of those you are using. You will not need to 'lose' or 'unshow' the fields, and can simply use **PAFUNC**(3) to clear the screen between the various sections if required. Note that if you use the **PAMENU** function the menu field must either be on a higher level than the **PAOVTH** value, or you must temporarily reduce the **PAOVTH** value for the **PAMENU** call.

Using Overlay Fields

If you use higher level numbers for help screens and other transient fields, you should set the **PAOVTH** value lower than the level of the field(s) you wish to display in this way and call the **PANELF** or **PANELS** functions as required. PANEL allows overlay fields 'on top of' other overlay fields, but if you do this and wish to use the *funsh* function to remove them you must remove the fields in the reverse sequence to that in which you displayed them. This will be obvious from a consideration of the 'save' technique described above to implement overlays.

Implementing a Pop-up Entry Window

PANEL provides an easy method of implementing a 'pop-up' data entry window. Such a window can contain as many fields as you like without being limited by the number of screen save pointers

described above. The technique is to define a single field the size of the window you require, on a non-zero level, and then define all the prompt and entry fields on a different, non-zero, level.

When your program needs to pop-up the window, the following steps are required:

- Set **PAOVTH** to a value lower than the level number of the single 'window' field and display it using **PANELF** with *fdisp*.

- Set **PAOVTH** to a value above the level number of the prompt and data entry fields. Set **PAPSLO** and **PAPSHI** to the same level number as the prompt and data entry fields.

- Use **PANELS** *pdisp*, *pshow* and *pread* functions to display and read the data entry fields. The fields will be displayed in your window, but 'overlay mode' will not be used, so there are no limits on the number of fields. 'Overlay mode' is not required at this time because the 'current screen contents' simply consist of a blank 'window'. (You can of course use **PANELF** instead of **PANELS**).

- When data entry is complete, call **PANELF** for the single 'window' field with a function code of *funsh*. The previous screen contents will be restored and the window and all the data entry fields in it will disappear.

This technique, with variations, can be used for a wide variety of data entry requirements. The 'window' can either be a pop-up area of a different colour, or could perhaps be the 'bottom half of the screen' in which different data is to be entered.

Defining overlay fields

The **panel** program uses a different technique for displaying level numbers than that described above. In effect you can 'visit' each level to bring that level's fields 'to the top'. This process continues until you return to level 0, at which time all the other levels are removed.

Using Levels and Overlay Fields

By sequencing your selection of level numbers you can arrange for the screen you are designing to look as if you had displayed the levels in the appropriate sequence for your application program. The level is selected simply by entering the level number at the main prompt. You can also change levels after selecting a field, prior to copying or moving it.

Earlier versions of **panel** could run out of memory when a large number of levels and extended attributes were defined, but the 8086 versions of **panel** are now compiled using a large data model (when available) to avoid this problem.

Testing screens with overlay fields

When you use **pantest** to exercise a screen which comprises several levels, you should be aware that the version on the distribution disks is linked with the default libraries which allow a maximum of 16 concurrently 'overlaid' fields. Since **pantest** sets **PAOVTH** to zero, this limit applies to the number of fields which can be shown on each non-zero level.

If this limit does not suit the style you have selected for level numbers, you can either recompile **PNLXASCC.C** and relink **pantest** or simply change the use of **PAOVTH, PAPSLO,** and **PAPSHI** in **PANTESTC.C** to reflect your use of levels.

The Development Cycle

Having seen how the programs and functions in the PANEL system perate, how do you actually use them to develop screen-based pplications? This section provides an outline of that process, which will need to be adapted to conform to your own development standards.

Planning

You need to determine at a fairly early stage which of your application screens will be dynamically loaded, and which will be compiled into the program. You are more likely to choose dynamic loading for menu and help screens which require little user interaction, and to choose code generation and linking for complex data entry screens. If you dynamically load such a data entry screen it is more difficult to process and program maintenance tasks may be harder. If your compiler has limits on static data, this may force you to use dynamic loading for some screens.

ou also need to plan the method you will use to define help boxes other than those associated with specific fields) and pop-up menus. In many cases the ten levels available for a panel will allow you to define all the associated fields of this type as part of the main entry panel. In some cases, however, the same pop-up menu will be required for several different entry panels, and will be better defined as a panel in its own right.

You may also need to set standards for the level numbers used in your application. Although **panel** can only define level numbers between 0 and 9 you may find it helpful to modify the generated levels inside your program to use more of the available range of 0 to 127. Remember that when you have multiple overlapping fields on the screen you have to remove them in the reverse sequence from that in which they were displayed, if you need to restore the screen below for further entries. If using different level numbers assists this task, then you may find it worth the effort. It is also dvisable to assign unique names for all your data entry fields across the application.

The Development Cycle

Screen Design and Programming

The next step is to define your screen panels in accordance with your plans. As each screen is designed, test it with **pantest** and note any new validation functions which may be needed in your application. These can be implemented by modifying the validation source files provided with the PANEL system and linking them with your application to replace the standard library versions.

After you are reasonably satisfied with each layout, you can generate code for the screen using **pangenc** and compile the resulting 'c' file.

Refinement of screen designs proceeds in parallel with development of the associated application programs. Each application program may include one or more of the generated screen header files and will be linked with the object files resulting from the compilation of the generated code. Your makefiles should reflect these relationships.

Building the Application

As the application nears completion, you can modify your code as indicated in the reference manual so that all associated dynamically loaded panels can be combined into a single file. This reduces the number of files required by your application and can also make loading more efficient.

Housekeeping

Remember to safeguard the panel files for your application along with your own source and header files. If you have used the multiple file option for dynamically loaded panels, you must keep all the separate panel files to enable you to rebuild the multiple file.

The PANEL User Interface

This section describes the way in which your users will use your pplication. You may wish to adapt some of the text to incorporate n user manuals for your programs.

In designing the PANEL user interface, the two main principles have been simplicity and consistency. The first impression that the user should get from the sight of the screen layout is a framework into which data can be entered and an immediate indication of how to enter the data. PANEL achieves this by encouraging the use of messages to the user which will appear at a constant, familiar, screen location, and by providing a software-controlled 'screen protect' facility which restricts input to the defined areas. By reading the data field by field, your program can also give immediate indication of errors.

The actual process of data entry is made easier by the inclusion of several data editing and control features:

- Cursor left and right functions permit non-destructive movement in the field, so that the operator can go back and correct a character without retyping the whole entry.

- The destructive backspace function operates in the same way as for command entries and clears the entry backspaced over. In a field initialised with fill characters, the fill character can be used to replace the deleted character.

- A character can be deleted, closing up the characters to the right.

- A blank can be inserted under the cursor, with movement of the remainder of the field to the right.

- A field line can be erased from the cursor position.

- Lines can be inserted or deleted in multiple-line fields. Cursor up and down functions are supported, and cursor wraparound occurs within the field in both directions.

- The TAB function moves along the field word by word.

- The use of the escape or 'program function' keys, and cursor movement out of the field, are reported to the application program to allow the appropriate action to be taken.

- A question mark or other 'help' key can be used to provide special 'help' information for the field.

User Keyboard Functions

Because the PANEL system allows so much flexibility in application design, particularly through the validation exits provided by the system, it is impossible to give hard and fast rules about the user interface and the operation of keyboard functions.

As an example, the 'carriage return' function is normally used on entry either to move from one line to another inside a field or to move from one field to the next. Using the exit provided at the end of field entry, your application program can arrange that 'carriage return' terminates the entry of a complete screen, so that the arrow keys have to be used to move from field to field when the field is not completely filled. This feature can be used if end-users are already familiar with this style of operation as a result of using other applications.

The following descriptions of keyboard functions assume that the 'default' function assignments have been used in your application.

Carriage Return

This function will move the cursor to the start of the next line during entry into a multi-line field, or to the next field in sequence when on the last or only line of the field.

Escape

The escape function causes an immediate exit from field entry, and will normally be assigned by the programmer to return to the previous application function.

Destructive Backspace

his function will backspace the cursor, erasing the entry ackspaced over. The erasing character will either be a blank space or the defined fill character, the latter being chosen when there is already a fill character at the current cursor position. The function is ignored in the first position of a field.

Cursor Left

The cursor is moved to the left non-destructively. At the start of a line, the cursor moves to the end of the previous line in a multi-line field, or to the previous field in sequence from the first or only line of a field.

Cursor Right

The cursor moves to the right non-destructively. The character already in the new position in the field is accepted without alidation. At the end of a line the line entry is only terminated if the 'CR Required' attribute is off.

Cursor Up

In a multi-line field, the cursor moves up to the same position on the line above, if present. On the first or only line of the field, the entry is terminated and the cursor returns to the start of the previous field in sequence.

Cursor Down

In a multi-line field, the cursor moves down to the same position on the line below, if present. On the last or only line of the field, the entry is terminated and the cursor moves to the start of the next field in sequence.

rase Line

The line is erased from the cursor position to the end of the line, with all characters replaced with the defined fill character for the field.

Insert Character

A blank is inserted at the position of the cursor, and all characters on the line to the right of the cursor move to the right. The rightmost character on the line is lost. This function is suppressed in a field for which an entry mask has been defined.

Delete Character

The character at the cursor position is deleted, and all characters on the line to the right of the cursor are moved left one position. The defined fill character for the field is placed at the right-hand end of the line. This function is suppressed in a field for which an entry mask has been defined.

Insert Line

Provided the cursor is at the start of a line, that line and all following lines of the field are moved downwards and a blank line is inserted at the cursor. The last line of the field is lost. This function is suppressed in a field for which an entry mask has been defined.

Delete Line

Provided the cursor is at the start of a line, that line is deleted and all following lines of the field are moved upwards. A blank line is inserted at the cursor. This function is suppressed in a field for which an entry mask has been defined.

Tabulate

The tab function will skip to the next 'word' in an entry field, which is defined as a non-blank character following a blank character. After the start of the last 'word', the cursor moves to the position following it, if available. In a field for which an entry mask has been defined, it moves to the next block of 'entry' characters. If there is no position in the field to move to as defined above, the cursor moves to the start of the next line in a multi-line field, or to the next field in sequence from the last or only line of a field.

Query

he query function is highly application dependent. If a 'help message' or 'help box' has been defined for the field, that is displayed until the next character is keyed. Otherwise, in the first position of a field line the key causes an immediate exit from field entry with a special code returned to the program; elsewhere in a field a question mark is displayed in the field.

Home

The home key operates on one of three levels. First it will return the cursor to the start of the current line. If already there, the cursor moves to the start of the field. And if it already at the start of a field, it will normally move to the first entry field on the screen.

Program Function Keys

Again, the action is application dependent. PF1 is normally used to end entry to a complete screen while retaining the entries already made.

Q and A

This section aims to explain and draw attention to some of the less often used PANEL features in the form of questions and answers. The questions are grouped only very loosely into categories, and there is considerable overlap of topics. Many of the questions are derived from real support calls.

(When a filename with a '99' is mentioned in the answers, replace the 99 with the value for your own interface).

Q. Why does PANEL come up on my IBM PC with underlines on every line?

A. You are using the colour tailoring parameters with a monochrome display. When using the memory-mapped versions, you can use a DOS *SET* command to set *PNLMONO* to any non-null value.

Q. My program is to run only on an IBM PC. Can I set up tailoring so that full colour will be used when available, but my program wil still run well on a monochrome system?

A. This happens automatically if you select the memory-mapped options for PANEL and your applications. If not, the most flexible method is to have your program check the equipment flags and then select the tailoring file required. You can create two copies of **PANEL.VDU** with different names, and pass the appropriate file name to **PAINIT**. Alternatively you can use **vdugenc** to create two separate source files and modify one by changing the name of the generated array. Then link them both into your program and use **PABM** to move the alternate file into **CPAVDU** if needed.

Q. The 'beep' on my PC is very loud when I am working late at night and PANEL seems to use it whenever I mis-key something. Can I turn it off?

A. Yes, simply use **tailor** to delete the '7' from the **BELL** screen function specification. For an IBM PC you can also place two values in this parameter which define a frequency and duration. Values of about 83 and 20 respectively correspond to the standard 'beep' and increasing the first value decreases the frequency. To

modify the memory-mapped version, change the PAIBEL values in the low-level module in the same way, either by recompiling the odule or assigning values at run-time.

Q. 'Boxes' round fields are defined for my system, but I cannot find the definition when I use tailor.

A. We have tried to use highlight type 14 (E) for boxes, and highlight type 15 (F) for any necessary graphics mode setting. This can be changed if required.

Q. I need two or three sets of tailoring parameters readily available, but the TAILOR.DAT file is too big to keep on my working disks. Can I cut out some of the entries?

A. Yes, but you need to know how to read in the file with the MS-DOS 'debug' program and then write it out again. First move the entries you need to the first few positions, then read the file in. To retain N entries, you need to write out (N+1) kilobytes using the w' command. The size needs to be placed in the CX register. efore writing out the file, place the value N in the eleventh byte of the file.

Q. It would be useful if the PANEL editor displayed the cursor coordinates in a corner of the screen. Could this feature be added?

A. It is there now, if you really need it. Just define a special PANEL.VDU file using tailor, with a cursor positioning sequence which first moves to a fixed position and outputs just the coordinates in fixed-length ASCII, then does the real positioning sequence. On the IBM PC, you can only do this if you use the ASCII driver.

Q. The highlighting on my terminal stopped working properly when I enabled the 'screen save' feature with a call to PASCSE.

. If your terminal uses attribute bytes then you must delete the highlighting sequences if you wish to use overlays or borders. PANEL does not support this combination.

Q. After using the PANEL editor to reshape a field on the HP150,

the highlighting for the old field sometimes remains on the screen. How do I get rid of it?

A. Display a help screen and then return to the panel display. Clearing the screen to do this removes the unwanted attribute bytes.

Q. I need many more key functions than PANEL limit, and I would like to define alternate keys so that more than one performs the same function.

A. You can do this provided that your keyboard does not generate any key sequences with trailing characters that have to be ignored. This enables you first to use the ignore table to extend the PF key definitions from 5 to 14. The special 0xff flag in the ignore table after the last entry indicates that it is being used in this way.

If you generate code for the **PAVDU** array using **vdugenc** you can extend the limit even further. First increase the size of the generated array and add the required pairs of key values at the end, followed by a 0xff flag. Then link the extended array with your program. Before calling **PAINIT** place a value in **PAKTS** which is 20 plus the number of bytes by which you have extended **PAVDU**, up to and including the 0xff flag. Increase PAKTS by one to make it an even number. The **PAKTS** value is used by **PAINIT** as the number of bytes from the start of the ignore table to scan for the 0xff flag.

To define alternate key functions, place code in **PAKVAL** to transform the passed value to that required. If you really need all those functions your own program must decide what to do with them.

The memory-mapped version can also use additional function keys by adding entries to the PAIKEY and PAIIGN tables in the low-level module. PAKTS should also be modified as described above.

Q. I plan to port my DOS applications to UNIX eventually. What steps should I take now to assist me in future?

A. You can use some of the techniques which you will find in the supplied source files, in conjunction with the PANEL portability header file **pnlc.h**. We do not guarantee that this will provide a universal panacea for portability problems, but we find that PANEL operates successfully in several different hardware environments. PANEL works extensively with single byte values between 0 and 255, and we try to remember to use the **UCI** macro whenever such a value is assigned implicitly or explicitly to a four- or two-byte integer. This provides good protection against sign extension problems and does not involve any overhead when masking is not necessary.

Q. How do I find out what field was being processed when the user hits escape during a screen read?

A. Look in the global variable **PAFLXS** for the field number.

Q. How do I start a full-screen read other than at the first field?

A. Set **PAFLX** to the field number before calling **PANELS**.

Q. My application needs to emulate a block mode terminal: I need to ensure that a specific function key is used to finish a screen read, rather than letting it finish after the last field has been entered. How can this be done?

A. Modify **pnlfvalc** to set up a global variable called **LAST**. In your program, after bringing a panel into use with **PAUSEP**, loop through the **PAFPT** array to count the number of input fields (PAFPT[i]->sa & input) and place this value in **LAST** (omit the loop if the number of fields will never change). Code a statement at the end of **PAFVAL** as follows:

```
if ((PAFLX == LAST) && (res < 11)) return(1);
```

The effect of this will be that when the last field has been completed, the cursor will move again to field 1 unless a function key has been used to end entry. **PANELS** will return the function

key number for analysis. Note that you may also wish to include some special processing in **PAFVAL** to cope with the escape key (res == 2).

Q. How do I make the colour of a field depend on its contents? I would like fields with a negative value to appear in red.

A. Do not be afraid to modify the field control block. When you are ready to display your screen simply move the appropriate highlight type into the fcb **hightype** field. Remember to reset it afterwards. There is also a **PANELF** call, **fhighc** which will change the highlight type of an already displayed field. Simply redisplaying it with a different highlight type is not portable to systems which may need to change attribute bytes on the screen.

Q. My screen layout has a rectangular matrix of fields in one area and I would like to be able to move between them at random, rather than in the linear mode provided by PANELS. How do I do this?

A. Place a switch statement in **PAFVAL** which is activated when the field number in **PAFLX** indicates a field in the required area. Then test the passed **res** value to determine the direction of exit from the field and return a value from the function which is **PAFLX** plus or minus the appropriate value to move to the selected field.

Q. I have several blocks of 'help' information to display one after another in the same area overlaying my main panel. How can I avoid having to 'unshow' each one before showing the next?

A. The best method is to set down one field using the 'overlay' technique and then display each successive help field on top of it using level zero (or adjusting **PAOVTH**) to avoid using the overlay feature. When the 'help' session is over, unshow the first field to reveal the previous screen contents. Think of this as laying down a rug on the floor on which you can place other objects. When you lift up the rug with everything on it, the floor is revealed again.

Q. When I dynamically load several panels I sometimes fail to get all the fields loaded. Why?

A. The **nf** field in the panel control block has two functions. It receives the number of fields in the loaded panel, and it can also be preset to a non-zero value to limit the number of fields to be loaded. If you use the same pcb more than once and fail to reset the **nf** field to zero before each load, you will never be able to increase the number of fields loaded.

Q. When I link my program I get lots of undefined symbols. Have you left out some modules?

A. Usually this problem is caused by naming files to the linker in an incorrect sequence. Most linkers search libraries in the order you specify them, and since the PANEL library uses standard C functions, it must always be specified ahead of all compiler libraries. In the UNIX operating system, ensure that all your object files precede the panel library on the command line.

Q. I compiled one of your validation source files and got a few warning messages. Is this wrong?

A. The PANEL source has been through over two dozen different compilers on six different processors. In the distributed source files we have tried to limit the number of conditional compilation statements to make the code more readable. Unfortunately this has meant that warnings appear with some compilers which we could only eliminate by changing the code in a way that another compiler would reject. If you are using the same compiler version as we did, the warnings you get will be harmless, unless you have modified the code to generate some new ones.

Q. How can I reduce the overhead for the floating point routines included in my program as a result of calls in the PANEL library?

A. The only module which makes floating point calls is **pnlnumvc**. If you wish to remove these, we suggest that you modify the code in the **numv** routine for currency fields (dp == 2) by replacing any decimal point in the input string with white space and using sccanf to find two (long) integers, the second limited to 2 bytes. Multiply

the absolute value of the first by 100 and add the second. Then transfer the correct sign to the currency result, which should contain a value in pence (or cents). The corresponding sprintf call will also need to be changed. You will end up with a more comple routine, if you allow fully for all the exception conditions, but you will avoid the overhead of floating point I/O operations. All numeric fields can use the same array of long integers, and the **PANUMD** array will not be required.

Q. Can I make numeric fields with a zero value display as blank rather than as '0.00'?

A. The provided numeric validation routines do this when the 'X' attribute is set.

Screen Design Guidelines

Enough books have been written about screen design, and the an/machine interface, to make it unnecessary to do more than rovide a check list of a few points worth remembering when you design your screen panel layouts. PANEL has facilities which enable you to design the layout in almost any way you wish, and most designers have strong views about which way is best. With apologies to the experts, here is our check-list:

- **Be consistent.** Have error and prompting messages always appearing in the same place. In a set of fields, either always require a carriage return to complete the field, or make each field the exact length and always switch to the next field automatically.

- **Be informative.** Never allow your application to throw up a screen which does not give the operator guidance about what to do next.

- **Do not be too informative.** It is a mistake to overload the screen with instructions which may cause an experienced operator to miss an important exception report. It is better to provide 'help' screens which may be called up easily in case of need.

- **Use highlighting in moderation.** Just because you have a screen which has seven different highlighting types does not mean that you have to use them all. The operator is looking for the easiest ways to enter data, not at how smart your hardware is.

- **Delimit fields.** It helps to reassure the operator if it is clear how many characters may be keyed into a field. Either place a fixed marker following the field, or if highlighting is not available use a field initialised with a string of non-blank characters.

- **Align fields.** An orderly column of data entry fields helps to minimise the operator's eye movements.

Screen Design Guidelines

- **Validate input early.** It can save unnecessary keying if you report on errors as soon as the program is able to detect them.

- **Use menus.** They help to give your application a logical and easily understood structure.

- Try not to intermix screen styles within an application. It is often an advantage to be predictable.

- Balance your screen designs and make them interesting to the operator. This is what the PANEL system is for!

Section II
Reference Manual

Introduction

This reference section contains separate subsections for each subject in the following categories:

- commands (programs)
- library functions
- file types
- global data areas and structures
- features

The subsections appear in a single alphabetic sequence by subject name.

NAME

Attributes - Field entry attributes

DESCRIPTION

The field entry attributes currently available are shown below. The descriptions refer to the 'standard' validation routines: note that in a specific application a non-standard validation module may redefine the significance of any or all of the attribute elements.

- **No input (N).** This indicates a 'fixed' field, which will not be included in any data record created from the panel, and which will not be processed by the 'read' functions.

- **Key Field (K).** The 'K' attribute is used by the PANEL data management program **pd** to identify fields which are to be used for data retrieval. Panel designs with up to ten key fields can be used by the data management programs, and currently they must be single line fields only. You may of course use the attribute for any other purpose in your own programs.

- **Upper Case (U).** All input to the field is to be converted to upper case as it is entered. The standard processing affects letters 'a' to 'z' only; for national alphabets in which other letters have to be converted to upper case it is necessary to modify the source code of the character validation routine as indicated in the comments.

- **Echo Input (E).** Input fields have this attribute by default. It can be deleted to provide protection for password fields.

- **CR Required (C).** Fields with this attribute require a carriage return or other valid function key to be entered before the entry for each line is considered complete. Without this attribute, entry ends when the line is full.

Attributes

- **Wipe clear the field (W).** This attribute causes the field to be erased to its 'fill' character as soon as any data entry is made, effectively preventing editing of the existing entry. The cursor right and tab functions can be used to 'bypass' the wipe attribute. It is most often used for numeric fields which are entered left justified and then displayed right-justified.

- **Right-Justify (R).** This attribute does not affect entered data until the field is completed, at which time the field contents are right-justified with left blank fill. Most often used for numeric fields.

- **Integer (I).** Data entered into a field with this attribute is checked as it is entered (by default 0-9, minus, or decimal point), and is also checked to ensure that a valid number has been entered when the field is complete. Exit from the field will be inhibited and the bell will sound if this is not the case. A valid field is redisplayed (right justified if the 'R' attribute is also set) and excess decimal places are deleted. The version of the validation module in use will determine the maximum input precision and whether decimals are rounded or truncated.

- **Currency ($).** Operates as for the 'D' attribute but redisplays the value with two decimal places.

- **(Undefined) (X).** This attribute is available for use by user supplied validation routines. It is also used as a modifier in the standard routines, either to permit a completely empty date field to pass the validation tests, or to cause a zero decimal or currency field to be displayed as blank.

Borders

NAME

orders - surrounding fields

DESCRIPTION

The panel display routines can display a 'box' as a border surrounding the field. Border processing only takes place when the 'B' attribute is specified for the field. The field is then displayed with the default or alternate border according to the display function used.

The default and alternate border characters are specified in **PANEL.VDU**, as set up by **tailor**. The first highlighting type found with a method code of 8 should have a special 'on' sequence to define the default border and a special 'off' sequence to define the alternate border. Each sequence consists of eight characters as follows:

- The first two characters define a highlighting type and method if such is required to display the graphics characters. When specified, the border highlighting for a specific field border cannot override these values, which always take precedence. The two characters should be zero if a special 'graphics mode' is not required.

- The next six characters define respectively the top left, top right, bottom left, bottom right, horizontal, and vertical characters. If fewer than six characters are specified, the last character is propagated to the right.

If no highlighting type with method code 8 is found, or the selected sequence is not defined, the 'box' defaults to all '+' characters.

The box definition for an individual field can be overridden by using the 'BD' extended attribute. The extended attribute string consists of up to six substrings delimited by any character. The delimiter character must start and finish the string. The first two substrings are numeric values to define the highlight type and modifier for the border. If not specified, these default to the same

values as the field highlighting. The third substring is up to six characters defining the border as described above. If this third substring consists of the single letter 'A', the alternate border is selected. The fourth substring is an optional 'legend' which is placed in the top edge of the border. The fifth and sixth substrings are numeric values to define the highlight type and modifier for the legend. If not specified these default to the same values as the border highlighting.

A negative value for the border highlight type in the first substring results in the display of the field being inhibited: only the border is displayed.

SEE ALSO

Extended Attributes

NOTES

The field border is not taken into account when determining the start position or size of the field with the field control block parameters: the border therefore lies 'outside' the field. When a field abuts an edge of the screen the whole of any defined border is suppressed.

When defining 'bordered' fields with **PANEL**, you can overlay a corner of another field with the border characters, even if the fields are defined at the same 'level'. If you subsequently move the 'bordered' field away, the overlaid corner will not be redisplayed automatically. To reshow the screen correctly, display a help screen and then return to redisplay the panel.

NAME

nvironment - for MS-DOS, PC-DOS, UNIX and XENIX

DESCRIPTION

Panel programs and library routines make the following use of environment variables:

VDU= is used to locate the **PANEL.VDU** information as described in **PAINIT.**

MSG= defines either a fixed message file name or, if a directory name, the directory to be used for message files. Message files are used by **panel, tailor** and **pangenc.**

VDU= is used by **panel, pantest, pangenc** and **pd** to specify a directory for panel description files.

TERM= is used to find **PANEL.VDU** information in **TAILOR.DAT** s described in **PAINIT,** and also by **paneltc.**

TAILOR= is used to find **TAILOR.DAT** as described in **PAINIT.**

PD= specifies a directory to be used for data and index files by **pd.**

PNLMONO= if non-null, is used to force selection of monochrome attributes when using the memory-mapped versions.

PAIATT= and **PAIBOR=** are used to specify hexadecimal strings of characters for attributes and borders in the memory-mapped versions.

The PANEL convention for environment specifiers is as follows: if the environment variable value ends with the path separator character, it is taken to be a directory name, and if not, it is taken to be a complete filename.

In systems which do not support an environment, processing proceeds as as if the environment variables were not defined.

Extended Attributes

NAME

Extended Attributes

DESCRIPTION

Extended field attributes provide a powerful and flexible means of defining entry and display parameters. Up to sixteen types of extended attribute may be defined. Currently code for processing seven of these is provided in the Panel subroutine library, and new types may be added by the application developer to meet specific validation requirements.

The type of extended attribute is defined by its position in the table of attributes, but each also has a two letter code which is defined on line 202 of **panel.msg**, and which will be used in the following descriptions. New codes can be added to line 202 of **panel.msg** as required.

Each extended attribute consists of a string of characters which has no inherent format. Some of the extended attribute types consist of sets of substrings which may be delimited by any character which does not appear in the substring. The delimiter character must immediately follow the two code letters and must also appear at the end of the string. The source file for **pnlxascc** must be modified slightly if more than 32 substrings are required.

The following routines are currently provided to support extended attributes:

HM - Help Message

The help message text is displayed on the current message line whenever the defined help key is depressed anywhere in the field. Note that the highlighting is 'borrowed' from the **PAMFCB** control block: if no **PAMSGE** call has been made the help message will not therefore be highlighted.

The help message remains on the screen until any key is depressed.

CL - Character List

The attribute string is used as a convenient means of overriding the **PAVAL** string containing the valid characters for the field. It is therefore only effective when Character Validation number 1 has been specified for the field.

PV - Picture Validation

This attribute string allows different validation to be specified for each field character by special codes which form a 'picture' of the field. The sample routine uses the picture characters *9*, *a*, *A*, and *X* only, but it is easy to add to or modify the processing so that, for example, upper case conversion is selected for certain positions only. Character Validation number 3 must be specified.

VL - Values List

The values list is used for three different types of field line operations, controlled by the character and line validation numbers:

Line validation number 2 checks the entered data against the substring list and rejects the field line if no matching entry is found. If a substring is shorter than the field length the excess in the field must be all fill characters to ensure a match.

Line validation number 5 uses the list to override the default "Yes" and "No" strings which are emitted when the first character of the field is typed. You should ensure that each substring begins with a different capital letter, or you may never see it emitted.

Character validation number 8 enables the operator to scan through a set of predefined values until the correct value is displayed. As coded, a space or plus sign moves forwards through the list, and a minus moves backwards. A null entry in the list allows normal data entry by the operator. The first substring is a dummy value consisting of a single upper-case letter indicating the element to select for initial display - 'A' for the first, 'B' for the second, etc. The **PAXVAL** routine called before the field is displayed will either display the string corresponding to the letter or, if the field already contains data, will set the code letter (if possible) to match the displayed data.

Extended Attributes

BD - Border Attributes

The processing for border attributes is described under **Borders.**

NF - Numeric Field

The processing for numeric fields is described under **Numeric Fields.**

HB - Help Box

Similar to the help message, the message specified in the extended attribute is displayed when the help key is pressed. If both **HM** and **HB** are specified, the **HB** is ignored. The first and second substring are numeric values defining the position of the top left corner of the help box. Since the help box has a border, you should not enter zero for either coordinate. The remaining substrings each define a line of the help text. The box will be made long enough to take the longest of these lines. The highlighing is, like **HM**, borrowed from the 'message' field.

HS - Spoken Help

The processing is described in the Amiga section.

SEE ALSO

Validation source files

NOTE

The **panel** program field initialisation function does not use extended attributes which modify data entry. The only extended attribute actually used during the operation of **panel** is the border definition.

NAME

keyboard - display key assignments

USAGE

keyboard

DESCRIPTION

The **keyboard** program reads the **PANEL.VDU** information from the standard place and displays the name from the terminal ID field together with the labels (comment fields) defined for each of the twenty keys defined in the key assignment table.

If more than the first five function keys have been assigned, this fact is noted, but there are no comment fields for additional keys. The program does not use the output code fields in **PANEL.VDU** and can be safely run from a terminal which does not match the current contents of the file.

Only the labels are displayed, not the codes assigned to each key function.

FILES

PANEL.VDU

Numeric Validation

NAME

Numeric Fields - Validation and processing

DESCRIPTION

Field Attributes

With version 6.20 we have changed the designation of the 'integer' numeric attribute to be 'I' instead of 'D'. This hopefully makes its characteristic more easily understood. (For users of previous versions of PANEL who wish to retain the 'D': search PANEL.MSG for each occurrence of '$' and change the adjoining 'I' back to a 'D'. No program change is required).

The two numeric entry attributes set using the *define* field option in **panel** are therefore now 'I' and '$' for 'integer' and 'currency' respectively. In the absence of any extended attribute for the field, these two parameters validate and reformat entered data with zero or two decimal places respectively, and make the data available to the application program only as an ASCII string.

Note that the validation is performed for C versions using the *sscanf* function. With some compilers, idiosyncracies in this function may prevent us from reliably rejecting multiple values in this field, such as can be created by using the 'insert blank' key function in the middle of a number. Also, the reformatting is performed by *sprintf* and we have noticed variations in rounding when more than two decimal places are entered. If these matters are important to you, study and if necessary modify the validation code in **PNLNUMVC.C**.

Without the *NF* extended attribute, the **PANELF** *flnop* function and the **PANELS** *prjnf* function will take any value already in the field data area and perform the validation operation on it, reformatting and if necessary justifying it.

In addition to the 'I' and '$' attributes, the 'X' (user) attribute for numeric fields will accept a blank field as zero and display a zero field as blank. Without this attribute a blank field is not accepted on input.

Note that if you use the 'right-justify' attribute for numeric fields you will often wish to use the 'wipe' attribute as well.

The NF extended attribute

The *NF* extended attribute is used to specify additional operations on numeric fields, and to arrange for the data to be returned to your application as a numeric value.

The extended attribute string is divided up into substrings using any convenient delimiter. The first substring, if non-zero, is the array index into an array of numeric values. There are two arrays, **PANUML** for integer fields and **PANUMD** for currency fields. These arrays are defined as *long* and *double* respectively in C versions. For the FORTRAN version the arrays are named **PANUMI** and **PANUMR** and are defined as integer*4 and real*8 respectively. When this substring is used, a read operation returns the data to the array element specified, and a **PANELF** *flnop* function or a **PANELS** *prjnf* function extracts the value from the array and formats it into the field data area.

The second and third substrings, if not absent or empty, specify the low and high values for a field range check. This validation is carried out on a read of the field.

The fourth substring is only used with the PANEL C interfaces. It is a *printf* format specifier which is used to override the default format. This provides an easy method of, for example, selecting a different number of decimal places for a value. Note that it may be possible to crash your program with a bad choice of format specifier.

Examples:

```
NF/1/          - use array element 1
NF/3/0/999/    - use array element 3, test for positive less than 1000
NF//0/         - test for positive
NF////%-1.4f/  - display with 4 decimal places
```

The fifth and sixth substrings are only for the PANEL MS-FORTRAN interface. The fifth substring defines for integer fields the minimum number of digits to be shown. For example a

value of 3 will format a 'one' as '001'. For real fields it specifies the number of decimal places.

The sixth substring can be entered as an 'E' (upper or lower case) t force the use of the 'E' format for display. Normally the 'E' format is used only if the field is too small for the number of significant digits in the value. The 'E' format is accepted for input of data to a real ('currency') field. Note that the exponent is shown by default with two digits. If you prefer three digits you have to recompile the **PNLNUMVF.FOR** module after making the changes indicated in the comments.

PAANSW

NAME

AANSW - Obtain answer to message

USAGE

```
#include <pnl3.h>
#include <pnl5.h>
int r;
char *s,*sm;
uint sat;
PAMSGE(sm);
r = PAANSW(s,sat);
```

DESCRIPTION

PAANSW reads a single-line field defined on the current message line **PAMLN**, ending at the x-coordinate placed in **PAMEN** by a revious call to **PAMSGE**. It is assumed that **s** is a pointer to the start of the area reserved in **sm** to receive the answer. The returned text is placed at **s**, and will therefore overwrite a default value (or blanks) displayed at the end of the message by the call to **PAMSGE**.

The **pamfcb** field control block is used for the read, and the size of its answer field is set to the current string length of **s**. The read is carried out by calling **PANELF(pread)** after temporarily setting the fcb attributes to the value given as **sat**. The passed **sat** value can therefore be used to control the attributes of the read, such as whether upper case conversion takes place.

PAANSW returns the value returned by **PANELF** for the read operation.

SEE ALSO

PAMSGE, PANELF

PACHCL

NAME

PACHCL - Change colours (IBM-PC and PC-DOS only)

USAGE

```
int clsattr, borderattr;
PACHCL(clsattr,borderattr);
```

DESCRIPTION

PACHCL changes the colour attribute to be used for subsequent clearscreen operations on the current display page to the attribute value given as **clsattr**, and changes the border colour attribute to be associated with the page to the value given as **borderattr**. If the active display page is the same as the displayed display page, the border colour is changed immediately.

The 'active' and 'displayed' display page numbers default to zero unless changed by a call to **PAPAGE**.

A subsequent call to **PAFUNC(1)** will reset all the attributes to the base values specified in **PANEL.VDU**.

SEE ALSO

PAPAGE, PAFUNC

NOTES

PACHCL is only implemented in versions of PANEL which can use IBM Personal Computer ROM BIOS calls or the memory-mapped I/O, and should only be used on a system including an IBM Color/Graphics Adapter or compatible equipment.

The attributes passed to **PACHCL** are the physical attribute bytes described in the IBM PC Technical Reference manual.

PACINP

NAME

ACINP - Read a keyboard character

USAGE

```
int c;
c = PACINP();
```

DESCRIPTION

The **PACINP** function returns a single character from the keyboard. The operation uses the lowest level of read operation available in the operating system. The **PACINP** function calls **CPAINP**, for which the source code is provided.

PACINP is designed for use in interactive applications: consequently the concept of end-of-file for the keyboard input is ot defined.

PACINP does not echo the character read.

NOTES

In environments in which PANEL buffers screen output, the **PACINP** function flushes the buffer before reading a character.

It is usually preferable to use **PAGETF** to read keyboard input.

PACOUT

NAME

PACOUT - Display character with highlighting

USAGE

```
char c;
PACOUT(c);
```

DESCRIPTION

PACOUT sends a character to the display after processing any highlighting operations which have to be carried out on a character basis. These comprise the setting of the high bit of each character, adding a value to each character, and sending a prefix string before each character.

If the 'screen save' feature has been enabled, **PACOUT** will ensure that each character written to the screen is saved with its highlighting parameters in the screen save buffer for use when a field is to be overlaid on existing data.

SEE ALSO

PASCSE

PACRF

NAME

ACRF - field control block for current field

USAGE

extern panfcb PACRF;

DESCRIPTION

The **PACRF** structure is used by **PANELF** field functions to store the current field details. It is a static area which may be used by user exit routines to obtain information about the field being processed.

The **PACRF** structure is available during processing of **PACVAL, PAFVAL, PAKVAL,** and **PAFVAL** exits. It is not available during **PAXVAL** exit, which is called before **PACRF** has been initialised.

ven if the field being processed has an 'abbreviated' field control block, the **PACRF** area contains a 'full' field control block, with the current values of the dynamically altered fields maintained on validation exit. In such a case, however, the status field **sa** continues to indicate an abbreviated format fcb.

The **PACRF** structure is automatically saved during a 'recursive' **PANELF** call for display from a validation exit.

NOTES

The structure definitions and the external definition for **PACRF** appear in **pnl3.h**

PACSTA

NAME

PACSTA - obtain keyboard status

USAGE

```
int r;
r = PACSTA();
```

DESCRIPTION

PACSTA is used to obtain the status of the keyboard input stream for operating systems which support this function. It returns a non-zero value if a character is waiting, and a zero value otherwise.

If no keyboard status is available from the operating system, PACSTA will always return a non-zero value; however in such cases pnli.h should define the pre-processor symbol NOKS and this shoul be used to avoid calling PACSTA.

NOTES

It is usually preferable to use PAGETF to read keyboard input.

PACURS

NAME

ACURS - position cursor

USAGE

```
int x,y;
PACURS(x,y);
```

DESCRIPTION

PACURS is used to position the visible cursor for reading input which is to be echoed from the keyboard to the screen.

The column and row coordinates x and y are based on values of (0,0) in the top left corner of the screen. If either value is larger than the defined physical screen dimensions in **PAECB** the maximum value is substituted.

EE ALSO

PAECB, PASBAD.

NOTE

In environments in which PANEL buffers screen output, the **PACURS** function flushes the buffer after sending the sequence.

PACVAL

PACVAL - validate input field characters

USAGE

int PACVAL(inch, dinch, ninch)
uchar *inch, *dinch, *ninch;

DESCRIPTION

PACVAL is a routine provided by the application program which is called from within **PANELF** for each data character. Source code for commonly required functions is provided.

On entry to the routine, ***inch** and ***dinch** both contain the character just read and ***ninch** is a null. All information currently known about the field being processed will be in the field control block **PACRF**.

The return value from **PACVAL** will be interpreted as follows:

0 The character is valid, store it in the field data area and, if **echo** is set, display it.

1 The character is not valid, sound the bell and ignore it.

2 The character is to be discarded.

3 The character is to be discarded, but still test for end of field as if a valid character had been entered.

PACVAL will process information returned in the function arguments as follows:

- ***inch** will be stored in the field data area.

- ***dinch** will be echoed to the screen.

- ***ninch**, if non zero, will be used as the next input data character. This is similar to 'ungetc', but if the field read terminates (eg on a full field) then the character is forgotten.

The validation routine may modify the field data area at **PACRF.datablock** but modification of the **PACRF** structure itself is not recommended and may be ineffective.

The functions in the provided source file implement the **upper** field attribute and then use the character validation number **PACRF.charvaln** as a switch variable to select the required validation process. Additional validation functions can be added either as variations to existing ones or as new cases in the switch.

PANELF may be called 'recursively' from within **PACVAL**, but only for field operations other than **fread**. If any other operation performed within **PACVAL** moves the screen cursor, it should be restored to its original position before returning. The provided routine **PARSTC** does this.

SEE ALSO

PANELF, PACRF, Validation Source.

NOTE

The provided routines use static arguments **PVINCH, PVDINCH,** and **PVNINCH** to simplify the declaration of the function pointer table. These variables are defined in **pnl4.h.**

PAECB

NAME

PAECB - PANEL Environment Control Block
panecb - PANEL Environment Control Block Structure

USAGE

```
#include <pnl3.h>
extern panecb PAECB;
```

DESCRIPTION

The **PAECB** structure defines the physical environment in which the PANEL library is operating. The structure is initialised during the **PAINIT** processing and may be modified by the application program.

The fields in the structure are used as follows:

TERMID contains a copy of the ten-character terminal or system identifier from **PANEL.VDU**. This string is not null-terminated.

MAXCOL contains the highest column number available.

MAXROW contains the highest row number available.

PAIDEN is an integer value corresponding to the interface/library in use. For C language interfaces this value is almost always defined at compile time.

PADSP defines the current displayable character set as an array of 256 bits. The bit is set if the corresponding character is displayable. Currently the array is initialised to the ASCII range 32 to 126 unless the **PANEL.VDU** file indicated 'IBMPC' as the first five characters of **TERMID**, in which case the range 1 to 254 is used instead. The function **PATDSP** is provided in **pnltdspc** as a portable means of testing bits in this array.

SEE ALSO

PANEL.VDU, PAINIT

NOTES

The structure definition and external definition for **PAECB** appear in **pnl3.h**

PAFLX

NAME

PAFLX - field index
PAFLXS - saved field index
PASET - table of active fields

USAGE

```
#include <pnl5.h>
int x, start, finish;
PASET[x] = '\1';
PAFLX = start;
...
finish = PAFLXS;
```

DESCRIPTION

PAFLX is the index variable used by **PANELS** for full screen operations. It is made available to the application program to enable full screen operations to be started at a specific field number. This is done by setting **PAFLX** to a value in the range 1 to 255 before calling **PANELS**.

For safety, **PAFLX** is always set to zero before exit from **PANELS**. In order to find the field number at which a full screen read was terminated, the application program can test the saved value in **PAFLXS**.

PASET provides another method of controlling the fields actively used by **PANELS**. Any field for which the corresponding value in **PASET** is non-zero will be excluded from **PANELS** operations. The application program is responsible for resetting any altered **PASET** values.

SEE ALSO

PANELS

NOTES

The external definitions for **PAFLX**, **PAFLXS**, and **PASET** appear in **pnl5.h**.

PAFPT

NAME

PAFPT - field pointer table
PANFL - current number of fields

USAGE

```
#include <pnl3.h>
#include <pnl5.h>
panfcb *fp;
int x;
fp = PAFPT[x];
x = PANFL;
```

DESCRIPTION

PAFPT is the pointer table used by **PANELS** to define the set of fields to be treated as a full screen. The table is normally initialised by calling **PAUSEP** but may also be modified by the application program.

PANFL contains the number of active fields in the field pointer array, and is also set by **PAUSEP**.

SEE ALSO

PANELS

NOTES

Since fields are numbered from one, and **PAFPT** is a standard C pointer array, be aware that **PAFPT[0]** will always be undefined.

PAFREE

NAME

PAFREE - free the space used by dynamically loaded panel

USAGE

```
#include <pnl5.h>
int ret;
panpcb *pcb;
ret = PAFREE(pcb);
```

DESCRIPTION

PAFREE frees the data and field control block areas used by a dynamically loaded panel (which has previously been loaded by **PALOAD**). Normally, the return value is always zero; however some versions of PANEL will return a value of 255 if the C library function can and does detect memory corruption on a 'free' call.

SEE ALSO

PALOAD

NOTES

Unpredictable results will occur if the data or fcb pointers have been modified since the **PALOAD** call, or if the addressed fcb was not dynamically loaded.

PAFUNC

NAME

PAFUNC - low level screen functions

USAGE

```
#include <pnl3.h>
int n;
PAFUNC(n);
```

DESCRIPTION

The **PAFUNC** function performs one of ten predefined screen functions for which the corresponding definitions are defined by **PANEL.VDU**. The function is selected by the value **n** as follows:

1 Initialise the screen and keyboard.
2 Reset the screen and keyboard to pre-initialisation defaults.
3 Clear the screen and home the cursor
4 Erase to end of line.
5 Backspace cursor non-destructively.
6 Sound bell or flash screen.
7 (undefined)
8 (undefined)
9 (undefined)
10 (normally undefined, except for HP150).

The function return value is undefined. Unused values of **n** may be used for an application-specific purpose if the appropriate sequences are set by **tailor**.

SEE ALSO

PANEL.VDU

NOTE

In environments in which PANEL buffers screen output, **PAFUNC** flushes the buffer after sending the sequence.

PAFVAL

NAME

PAFVAL - Field validation

USAGE

```
int PAFVAL(fres)
int fres;
```

DESCRIPTION

PAFVAL is a routine provided by the application program which is called from within **PANELS** after each field has been read. Sample source code is provided.

On entry to the routine, **fres** contains the **PANELF** return value from the field just read. **PAFLX** contains its field number and **PACRF** is its field control block.

The **PAFVAL** function is used principally to modify the route taken through a panel by a complete panel read operation. The function should return zero if the route is not to be modified and the number of the field to jump to otherwise. For example, to skip a field if the current field contains an 'N', test the first data character ***PACRF.datablock** and then conditionally return **PAFLX-2** if **fres** indicates forward motion through the panel.

SEE ALSO

PANELS, PACRF, Validation Source.

PAGETF

NAME

PAGETF - get function or data key

USAGE

```
#include <pnl3.h>
int keyfunction;
uchar character;
uint res;
res = PAGETF();
keyfunction = res & 0xff;
character = (res >> 8) & 0xff;
```

DESCRIPTION

The **PAGETF** function is the normal method of reading and analysing keyboard input. It returns two values to identify a particular keystroke, having (waited for and) read keyboard input as necessary. If the keystroke is identified as a defined function key, its number is returned to **keyfunction** as follows:

1	CR	7	DOWN	13	TAB	19	PF4
2	ESC	8	EEOL	14	QUERY	20	PF5
3	BS	9	INS CHAR	15	HOME	21-	PF6-
4	LEFT	10	DEL CHAR	16	PF1		
5	RIGHT	11	INS LINE	17	PF2		
6	UP	12	DEL LINE	18	PF3		

If the **keyfunction** value is zero then the **character** will contain the returned data character. Both values zero indicates an undefined multiple-key sequence.

SEE ALSO

PACINP, PAINIT, PANEL.VDU

NAME

PAHION - send initial highlight sequence
PAHIOF - send final highlight sequence
PAHTYP - set current highlight type/method

USAGE

```
#include <pnl3.h>
int modifier;
int hightype, highmethod;
PAHION(modifier);
PAHIOF();
highmethod = PAHTYP(hightype) & 0x7f;
```

DESCRIPTION

These three functions are used to implement the portable highlighting used by PANEL. Application programs which perform highlighting on a field basis will not normally need to call these functions.

The **PAHTYP** function causes the **hightype** value to become the current highlighting type. The passed value must be in the range 0 to 15, or the return value will be undefined. The function returns the corresponding highlighting method value as defined in the **PANEL.VDU** file. If the high bit of the method byte is set, the current highlight type is that specified for use by the PANEL system programs as the default means of highlighting fields.

PAHION processes the sequence defined as the 'on' sequence for the current highlight type, as modified by the passed **modifier** value. Depending on the definition of the 'on' sequence special codes, the sequence may not be sent immediately but may instead be prepended to each following character sent by **PACOUT**. The use of the modifier is also controlled by special codes in the 'on' sequence. **PAHIOF** sends the 'off' sequence defined for the current highlight type, if any. The current highlight type and method are not changed.

PAHTYP

SEE ALSO

PANEL.VDU, PACOUT

NAME

PAINIT - initialise PANEL routines
PAKTS - keyboard table size

USAGE

```
include <pnl3.h>
int ret;
char *s;
ret = PAINIT(s);
```

DESCRIPTION

The **PAINIT** function initialises the PANEL run-time subroutines and for non-memory-mapped versions reads the **PANEL.VDU** data from disk. The data is only read if the first byte of the **PAVDU** area is null, which will be the case if a module created by **VDUGENC** has not been linked with the program.

The search strategy for the **PANEL.VDU** data is as follows:

1. If a file generated by **VDUGENC** has been linked, the **PAVDU** area is assumed to contain the data and no disk access is made.

2. If the string **s** is non-null, it is assumed to contain a full path and file name which is used to attempt to read the **PANEL.VDU** information.

3. If the string **s** is null, the VDU environment specifier is read, if present. If this indicates a directory name, the file **PANEL.VDU** is read there; if a full filename, this file is assumed to contain the **PANEL.VDU** data.

4. If no VDU environment entry is present, a **PANEL.VDU** file is sought in the current directory.

5. If no data has been found after step 4, the TERM environment specifier is read. If present, its value used as

a search key in the **TAILOR.DAT** index and if found the appropriate block is read from the **TAILOR.DAT** file. The **TAILOR.DAT** file is assumed to reside in the current MS-DOS directory, or as /etc/TAILOR.DAT for UNIX or XENIX operating systems. This assumption may be overridden by specifying a full filename as the TAILOR environment variable.

The return code from **PAINIT** is zero for a successful operation. A return value of 1 indicates a failure to open the data file and of 2 a failure to read it. Any subsequent call to **PAINIT** simply returns the same value as on the first call.

PAINIT sets up the **PAECB** structure and also analyses the key function table to determine how many function keys are in use. It does this by searching for a flag of 0xff in the keyboard ignore table which will indicate that this table is being used (in the entries preceding the flag) as an extension of the key function table. The **PAKTS** value (default 20) may be increased before calling **PAINIT** to increase the scope of the search for the special flag. This will only be necessary when a **PAVDU** area greater than 512 bytes has been defined by modifying the output of **vdugenc**.

In versions of PANEL supporting IBM Personal Computer ROM BIOS calls, **PAINIT** enables such calls if the first five characters of the TERMID field are 'IBMPC'.

In versions of PANEL for which no keyboard status is available **PAINIT** may modify the keyboard recognition strategy if it has been set up on the assumption that status will be used. In particular a single depression of 'escape' may be changed to require two depressions.

SEE ALSO

PANEL.VDU, TAILOR.DAT, PAECB, VDUGENC

NAME

PAKVAL - validate input field key functions

USAGE

```
int PAKVAL(kp)
uint *kp;
```

DESCRIPTION

PAKVAL is a routine provided by the application program which is called from within **PANELF** for each key function. Source code for commonly required functions is provided.

On entry to the routine, ***kp** contains the key value as returned from **PAGETF**, which may be either a data value or a key function, coded as shown in the source file. All information known about the field being processed will be in the field control block **PACRF**.

The return value from **PAKVAL** will be interpreted as follows:

0 The function is valid.

1 The function is not valid, sound the bell and ignore it.

2 Ignore the key completely.

The **PAKVAL** routine may also change the key value function number in ***kp**.

The validation routine may modify the field data area at **PACRF.datablock** but modification of the **PACRF** structure itself is not recommended and may be ineffective.

The functions in the provided source file implement support functions for character validation by using the character validation number **PACRF.charvaln** as a switch to select the required process. Additional validation functions can be added either as variations to existing ones or as new cases in the switch.

PAKVAL

PANELF may be called 'recursively' from within **PAKVAL**, but only for field operations other than **fread**. If any other operation performed within **PAKVAL** moves the screen cursor, it should be restored to its original position before returning. The provided routine **PARSTC** does this.

SEE ALSO

PANELF, PACRF, Validation Source.

NOTE

The provided routines use a static argument **PVKVAL** to simplify the declaration of the function pointer table. If you change this value it is not currently saved in ***kp**. This variable is defined in **pnl4.h**.

PALOAD

NAME

PALOAD - Dynamic load of panel file

USAGE

```
#include <pnl5.h>
int ret,full,data;
panpcb pcb;
ret = PALOAD(&pcb,full,data);
```

DESCRIPTION

The **PALOAD** function loads a panel's details from disk. The **pcb.pf** field in the panel control block must have been initialised with the panel name to be loaded and **pcb.nf** must either contain zero or the maximum number of fields to be loaded. A value of zero implies no limit. **PALOAD** fills in values in **pcb** with the 'nformation just input and sets up field control blocks and, optionally, data areas for the panel.

Any filename extension in **pcb.pf** is removed and replaced by '.pnl' for normal load operations. However if **PAMULT** equals one, then **PALOAD** will read the panel data from a multiple panel file which has been opened by **PAMLOP** and positioned by **PALOOK**. Note that **PALOOK** requires a panel name without extension in the **pcb.pf** field.

If **data** is non-zero then panel data areas are set up. If not, only the control information is loaded, presumably to allow the calling program to determine what the panel consists of without the intention of displaying it. Panel data areas are created by dynamic memory allocation and begin at **pcb.da**. The data area for each field is followed by a single null. Note that this differs from the method used when code is generated for the panel: in that case each ield line is followed by a null.

If **full** is non-zero then full field control blocks are created for each field; otherwise, abbreviated field control blocks are used. The abbreviated fcb does not include the field name.

PALOAD

The return code **ret** has the following significance:

0	Successful load
1	Unable to open file
2	Unable to read file
254	Out of memory
255	File corrupted

SEE ALSO

PAUSEP, panpcb, PAMULT

NOTES

PALOAD does not call **PAUSEP**. The calling program needs to do this to initialise PAFPT before making any calls to **PANELS** for full screen functions.

WARNING: If you re-use the panel control block to load another panel, remember to reset **pcb.nf** to zero first. If you fail to do this you will never be able to increase the number of fields loaded.

PALVAL

NAME

PALVAL - validate input field line

USAGE

int PALVAL(code)
int code;

DESCRIPTION

PALVAL is a routine provided by the application program which is called from within **PANELF** for each field line. Source code for commonly required functions is provided.

On entry to the routine, all information currently known about the field being processed will be in the field control block **PACRF**. The provided source file includes code to extract the current line using the fields in **PACRF**.

The return value from **PALVAL** will be interpreted as follows:

0 The line is valid and has not been altered in this routine.

1 The line is valid but needs to be redisplayed.

2 The line is not valid, but has not been altered in this routine: sound the bell.

3 The line is not valid, and needs to be redisplayed before sounding the bell.

The validation routine may modify the field data area at **PACRF.datablock** but modification of the **PACRF** structure itself is not recommended and may be ineffective.

The **code** value indicates whether the field is being called from the 'lineoperation' or the 'read' function of **PANELF**. A value of zero indicates that input has to be validated and reformatted, and a value of one indicates that data should be prepared for display only. The **code** may also have a value of -1 as described under **vertical scroll.**

PALVAL

The functions in the provided source file implement the standard numeric and right justified field processing as well as date validation, by using the line validation number **PACRF.linevaln** as a switch variable to select the required validation process. Validation functions can be added either as variations to existing ones or as new cases in the switch.

PANELF may be called 'recursively' from within **PALVAL**, but only for field operations other than **fread**. If any other operation performed within **PALVAL** moves the screen cursor, it should be restored to its original position before returning. The provided routine **PARSTC** does this.

SEE ALSO

PANELF, PACRF, Validation Source.

NOTE

A number of fields used by specific line validation routines are defines in **pnl4.h.**

PAMENU

NAME

PAMENU - process vertical pop-up menu

USAGE

```
panfcb *fp;
int ret;
int start,menht,menhm;
ret = PAMENU(fp,start,menht,menhm);
```

DESCRIPTION

PAMENU displays a multi-line field and moves a cursor bar from line to line to allow selection by the user. The field is displayed in 'overlay mode' and the normal display cursor is suppressed, if possible, during entry.

The **menht** and **menhm** values are used as the highlight type and modifier to highlight a single line of the field, starting at line number **start**. In this function only, lines are numbered starting at one in the field, but a start value of zero still selects the first line. The cursor up and down keys are used to move the highlighted line up and down the field, and then the carriage return key selects the highlighted item.

If a letter is keyed which matches one of the (upper case) characters at the start of a line in the field, that item is selected and the selection process terminates.

Before returning to the calling program, the function erases the menu from the screen and restores the previous displayed data. **PAMENU** returns -1 if the parameters were invalid, 0 if escape was keyed, and otherwise the selected line number.

NOTES

The source code for **PAMENU** is provided in **pnlcv09c**. An example of a menu appears in the **DEMO** panel and the function is called by **testc**.

PAMENU

The function can be adapted to process other shapes of menu field by using the same principle of setting up a dummy field for the highlighted item using values from the passed fcb.

For proper operation, the 'screen save' feature must have been enabled with **PASCSE**, and there must currently be two free save area pointers (see **Using levels and overlay fields**).

NAME

PAMSGE - Display message on message line
PAMEN - End of last displayed message
PAMFCB - Message field control block
PAMHD - Message Heading string
PAMLN - Message Line

USAGE

```
#include <pnl5.h>
char *s, *h;
uchar hm;
int i,j;
strcpy(PAMHD,h);
PAMLN = j;
PAMFCB.highmodi = hm;
PAMSGE(s);
i = PAMEN;
```

DESCRIPTION

The **PAMSGE** function displays the message **s** on the current message line **PAMLN**. If the string **PAMHD** is non-null then it is prepended to the string **s** before display, separated by a colon and a space. The x-coordinate of the message end point is placed in **PAMEN** for possible use by **PAANSW**. The function return value is undefined.

PAMLN is normally initialised by **PAUSEP** to the value defined for the panel in use, but may be changed by the application program as required.

The field control block used for messages and for **PAANSW** is **PAMFCB**. On the first call the attribute table in the **PANEL.VDU** information is scanned to find the first highlight type for which the method code has the high bit set; this value is used as the message highlight type. With memory-mapped versions which do not use a method code, the highlight type 1 is selected.

PAMSGE

If the message string **s** is empty then the message line is cleared. Passing a null pointer results in the previously displayed message (saved at **PAMFCB.datablock**) being repeated. If the combined length of **PAMHD** and **s** is too long for the screen an attempt is first made to remove any part of **s** preceding the first occurrence of '['. This effectively retains only a part of **s** enclosed in square brackets. If there is no '[' or if the string is still too long it is truncated on the right.

SEE ALSO

PAANSW, PAUSEP

NOTES

The external definitions for **PAMLN, PAMEN, PAMHD** and **PAMFCB** appear in **pnl5.h.**

The processing described above for null string pointers and empty strings has changed in PANEL version 6.20.

NAME

PAMLOP - Open multiple panel file and index
PAMLCL - Close multiple panel file and index
PAMULT - Multiple file control switch
PALOOK - Search multiple file index

USAGE

```
#include <pnl5.h>
extern int PAMULT;
char *multfile, *index;
int ret, full, data;
ret = PAMLOP(multfile,index);
. . .
pcb.nf = 0;
ret = PALOOK(&pcb);
ret = PALOAD(&pcb,full,data);
. . .
ret = PAMLCL();
```

DESCRIPTION

These functions allow panel details to be dynamically loaded from a file containing multiple panel descriptions. Such a file is created by concatenating individual panel files and an index for it is created by using **pnlindx**. The functions are designed so that separate panel files may be used during initial application development with **PAMULT** set to -1: this suppresses all 'multiple file' operations, but the calls may be coded ready for use later in a production version.

The **PAMLOP** function opens a file containing multiple panel details together with its corresponding index file (created by **pnlindx**). The passed file names **multfile** and **index** are used unchanged and file extensions have no special significance. The function returns zero to indicate success; return values of 1, 2 or 3 indicate respectively an open failure, read failure or memory allocation failure. **PAMLOP** does nothing (and returns zero) if **PAMULT** contains -1; otherwise it sets **PAMULT** to a value of one

to indicate that a multiple file is in use. Only one multiple file at a time may be used.

PALOOK is used to determine the seek position in the multiple file for the start of the panel named in **pcb.pf**. The panel name should have no file extension. It only does this if **PAMULT** contains a value of 1, that is if a multiple file is in use. The call to **PALOOK** may therefore be placed immediately before a **PALOAD** call and will only be effective when required. **PALOOK** returns 1 if the name is not found in the index; zero indicates success or no-operation.

PAMLCL closes the current multiple file (if in use) and frees the space used to maintain the index. The **PAMULT** value is changed from 1 to 0, effectively reverting to operation with separate panel files.

SEE ALSO

PALOAD, pnlindx

NOTES

The external definition for **PAMULT** appears in **pnl5.h.**

PALOOK uses a simple linked list to find the panel name, and always begins at the start of the list. If you have a large number of panels in a multiple file, and need a faster method, replace **index, PAMLOP,** and **PAMLCL** with your own procedure. To force **PALOAD** to seek before starting to read, first set **PAMULT** to a value of 1 and then place a record number in the global integer **PASEEK.** The record size used for panel files is 32 bytes. **PALOAD** omits the close of its input file when **PAMULT** is 1.

NAME

anel - screen panel editor
paneli - screen panel editor (memory-mapped)

USAGE

panel [*-z*] [*panelname*]

DESCRIPTION

panel is the program used to create and edit screen layouts. It displays an image of the screen and allows fields to be set up and manipulated interactively. A list of the commands appears below, and extensive on-line help can be displayed by keying the defined help key.

The *-z* option is required to load panel descriptions created by **anel** versions earlier than 6.00. If no panel name is entered on the command line, **panel** will prompt for one. Multiple separate panels can be processed in one invocation of **panel**, but only the name of the first panel can be entered on the command line. The name entered may be that of an existing panel or of one which is to be created.

panel controls up to ten separate levels of fields in one 'panel'. The levels are numbered 0 to 9. Fields may be any rectangular shape and those on any one level may not overlap. You can change from one level to another by keying the level number at either the main edit prompt or the field selection prompt: fields can therefore be moved from one level to another by changing level after selecting a field. Up to 255 fields may be defined for a panel.

panel will continue to display any fields defined at each level you 'visit' until level 0 is selected, at which time all the higher level fields are temporarily erased. The program will frequently do a complete screen repaint when fields are manipulated at a non-zero level number. Note that a full repaint is <u>not</u> required when using the PANEL subroutine library to display overlaid fields. A screen repaint works up through the level numbers, displaying higher level

numbers last. If you wish to view overlapping fields with a different priority, go down to level zero and then 'visit' the higher levels in the appropriate order. The current level fields are always redisplayed 'on top'. In general it is better to organise your screen design so that fields at higher levels will overlay those at lower levels.

The **panel** commands are normally selected with a single (initial) letter. The basic method of operation is to move the cursor to the required position and key the command letter. A list of the commands appears below:

Abandon Cancel a new field or cancel the selection of an existing field. The 'A' may be keyed with the cursor at any position on the screen.

Attributes Define field entry and display attributes. The attributes are specified in five stages: first the entry attributes and other code letters, then the character and line validation numbers, and finally the highligh type and highlight modifier.

Begin Start a new field or reposition the start of a selected existing field. To start a new field, the cursor must not lie within another field at the same level. A field may not be reshaped if you have changed to another level after selecting it. New fields are given a default name and attributes.

(block) The insert and delete character and line keys perform a block move operation on all fields at the current level which lie wholly to the right and below the current cursor position. An attempt to use these functions may be rejected if a field lies partially within this 'block move area'.

Copy Make a copy of a selected field. The new field is positioned on the current level at the current cursor position. A new default name is allocated, but all the other details are copied. You are asked to confirm whether extended attributes are to be copied also.

(cursor) The cursor may be moved at will over the panel, using the defined 'arrow' keys. Space acts like cursor right. Tab is the equivalent of eight cursor right entries. Home moves the cursor to the start of the line and then to the top of the screen.

Define Select field for option definition. After selecting this option a further menu is presented allowing field naming, initialisation, and specification of normal and extended attributes. Only the selected field will appear on the screen.

Delete Delete the selected field. The selected field is deleted, even if the cursor has been moved to another position before keying the 'D'. Confirmation is requested.

Edit Return to edit a screen layout after writing it to disk or after merging another panel definition.

nd Define the end position of a new field or reposition the end of a selected field. The end position must not cause an overlap with fields at the same level.

(escape) Used to exit from editing mode or to terminate entry of extended attributes.

Fill Define the fill character for a field. A blank fill character displays as '^' to make it visible. A numeric value may be entered at the prompt to define a non-typeable character, but this will bypass the 'displayable' check and may smash the display if you enter a 'control code'.

(help) Display help information. Help is context-sensitive and displays details of the currently valid commands.

Info Display information about a field or about the panel. If the cursor lies within a field, its parameters are displayed; otherwise information about the panel such as the number and total size of fields, and the levels in

use, is displayed.

Init Define initialisation for a field. During initialisation any defined attributes are enabled, so that for exampl if you have deleted the 'echo' attribute your initialisation will not be visible. An entry mask can be defined by including '^' characters in the initialisation to mark the entry positions. Non-mask fields are set to 'no-input' when initialised for the first time. Initialisation is deleted by entering the 'erase line' key at the start of each field line.

(level) Change level. You can change to another level in the range 0 to 9 by keying the level number at the main edit prompt or the field selection prompt. The level number is shown on the prompt line for levels other than zero.

Merge Combine another panel layout with the current one. Any overlapping fields will be ignored, with a warnin message. Duplicated field names will be changed to a default name, again with a warning message.

Move Move a field to a new position. The field is moved to the current level at the position of the cursor.

Name Define a field name for a field. The name must start with a letter and be different from the panel name and any other field names. By default, only fields for which you have defined a name appear in the generated header file for the panel which is included in your application program.

New Start processing another panel layout. You will be asked for confirmation if you have not just written out the current panel details.

Prompt Reposition the prompt line. The prompt line is moved to the current cursor line, which does not have to be clear of defined fields.

Quit Terminate the program. You will be asked for confirmation if you have not just written out the panel details.

Select Select a field for movement, reshaping, or deletion. The field will be highlighted if possible.

Write Write panel details to disk. You will be prompted to change the name of the panel if required, and asked to provide a comment or note. You must also select Horizontal, Vertical, or Unchanged sequence for the field sort. This determines whether **PANELS** will process the fields from left to right or from top to bottom.

Xtend Define extended attributes for a field. The display is replaced with a box which is used to enter extended attributes for the field. The first two characters of each attribute must correspond at all times to one of the valid code pairs. This may require you to delete with the 'erase line' key any line for which you need to change the code letters. Lines can be continued by entering a special code of '..' in the first two columns.

SEE ALSO

.PNL files, Attributes, Extended Attributes

PANELF

NAME

PANELF - perform panel field function

USAGE

```
#include <pnl3.h>
int ret, func;
panfcb *fp;
ret = PANELF(func,fp);
```

DESCRIPTION

The **PANELF** function is the single entry point for all of the panel field-level functions. It performs the function **func** on field **fp**. Names for the numeric values if **func** are defined in **pnl3.h** and are also used below.

(Code 0) — No action results for function code zero.

fdispu (Code 1) — Display the field on the screen. No highlighting will be carried out even if specified; however the border and overlay functions operate as described for **fdisp**.

fdisp (Code 2) — Set up any highlighting for the field and then display the field. After displaying the field any necessary sequence to turn off highlighting is sent (for method 4 only). This function is the standard one used to display a field by PANEL.

If a field has a border defined, this function displays it. In addition it tests to see if the field's overlay level **so** exceeds the **PAOVTH** threshold value and if screen save is enabled: if so the current contents of the screen are saved in a buffer allocated at the first available field save pointer location **PAFSVA**. After a successful save operation the pointer index number is saved in the field control block **so** area with its high bit set. This

indicates that the field is currently overlaying saved data.

fdispa (Code 3) — This function operates exactly as the **fdisp** function except that the alternate border is selected instead of the default border. If a border has been defined in the field's extended attribute area then there is no difference between **fdisp** and **fdispa**.

fbordd (Code 4) — This function should only be used on a field already displayed with the alternate border. It redisplays the border only as the default border. It is ineffective if the field's extended attributes specify an override border.

fborda (Code 5) — This function should only be used on a field already displayed with the default border. It redisplays the border only as the alternate border. It is ineffective if the field's extended attributes specify an override border.

funsh (Code 6) — This function is used to 'unshow' an overlaying field, that is to redisplay the saved screen data which was previously 'underneath' it. The buffer containing the saved data is freed. *IMPORTANT NOTE*: if the calling program fails to 'unshow' overlapping fields in the reverse sequence to that in which they were defined, the screen contents will not be correctly restored.

flose (Code 7) — As an alternative to 'unshowing' the field, the saved screen contents can be 'lost' with this function. The buffer area is freed, but the screen is not redisplayed. If the application decides to clear the screen while overlaid fields are displayed, it should call this function for each field first.

freadu (Code 11) — Read the whole field and return the PANEL function return code shown below. No highlighting will be carried out even if specified.

freadh (Code 12)
Set up any highlighting for the field and then read the field. If **PARDHI** is non-zero, it will be taken as a highlight type for the field which is to be used during the field read operation. If **PARDHI** is zero, the field will not be re-output by this function, so a previously unhighlighted field will not initially be highlighted unless the method code involves attributes which cause the appearance of the field to change. Characters will normally appear highlighted as they are entered. This function is the standard one used to read a field by PANEL.

freads (Code 13)
Read the field from the starting position specified by **fi** and **fj**. This function is normally used to restart reading a field from the point at which an escape or PF key was detected, if required. This function will not use highlighting. To do this it is necessary to (1) save **fi** and **fj**, (2) call function 21, (3) restore **fi** and **fj**, (4) call function 13, and (5) cal function 22. Note that in a field read under mask, the **fi** value is ignored and reading commences from the first entry position in the mask.

fhigh1 (Code 21)
Set the highlighting for the field. This operation involves writing the attributes in the appropriate positions if necessary, setting up indicators to cause the appropriate character-by-character operations to take place (as specified in the attribute-on sequence), and in the case of method code 4 issuing the appropriate attribute-on sequence. Note that special processing is performed when a field requiring attributes abuts the edge of the screen, and the positions of other fields may affect the success of the attempt to highlight the requested field. It should be unnecessary to use this function if the application only uses functions 2 and 12, which call this function internally.

fhigh2 (Code 22)
Complete the highlighting of the field. This function must be called after the operation which followed a call to function 21, in case a termination

sequence has been specified (method 4 only). This function is called automatically by functions 2 and 12.

fhighc (Code 23)
Change the highlighting of an already displayed field. The previous highlight type must be placed in **PAOLDH** in order to ensure successful operation. In the case of a field previously using attribute bytes, this function may need to write out blank characters to replace them on the screen. This function may also redisplay the field if this is necessary to change its appearance as requested. Currently, this function supports only highlighting methods 1, 3 and 4.

ferase (Code 24)
Erase a field from the screen. This function tests the specified highlighting and if necessary will write out blank spaces to replace attribute bytes. The contents of the field's data area must be initialised to blank characters prior to calling this function. Currently, only fields whose highlighting uses methods 1, 3 or 4 (or without any highlighting) can be erased with this function.

fsetm (Code 30)
This utility function will set up the mask field **sm** for the field by reference to the characters '^' currently found in the field's data area (first 16 characters only). The '^' characters indicate positions in which future data entry will be permitted. This function may be used by an application to dynamically change a field mask. (Alternatively you may assign new values to **sm** directly). If exceptionally this function is called for a multi-line field, only the last line of the field will be used to determine the mask.

ffilm (Code 31)
This function will fill the field with the defined fill character under the control of the entry mask. This function must be used to clear masked fields in preference to function 32 which will also erase the fixed characters in the field. If the field is not

masked, or if the RJ, numeric or currency attributes are set, then this function is equivalent to function 32. The function only affects the field's data area, and no data is displayed.

ffill (Code 32) — This function clears all of the characters in the field's data area which also lie in the displayed screen area to the defined fill character.

funfil (Code 33) — This function changes any characters in the field's data area to blank which were previously equal to the field's defined non-blank fill character. *WARNING:* in a numeric field which has '.' as fill character, the decimal point, if any, will also be cleared to blank.

fpadf (Code 34) — This function pads each line of the field with the defined fill character, replacing all blank or fill characters after the rightmost significant character.

flnop (Code 35) — This function performs any defined 'line operation' on each line of the field. The effect depends on the content of the user's **PALVAL** subroutine, which is called to perform the operation, and on the defined **linevaln** for the field. The **PANELF** return code is the return code from **PALVAL**, with the value for a multi-line field determined by a logical 'or' of all the individual results. This function is normally used to ensure that data pre-inserted into input fields is formatted with the correct number of decimal places and justification before display. If the NF extended attribute has been specified with a valid numeric field index number, this function takes the numeric variable and formats the value in the field data area.

ffild (Code 36) — This function clears the entire data area for the field to the defined fill character. If the data area width is one larger than the displayed width, the additional byte at the end of each line is set to a null string terminator.

The values returned by the PANELF function after a read operation are as follows:

0	Normal read, data has been entered.
1	Normal read, only carriage return was keyed on each line of the field.
2	Escape function keyed at any point of the field.
3	Cursor left keyed from the home position of the field.
4	Cursor up keyed on the top or only line of the field.
5	Cursor down keyed on the bottom or only line of the field.
6	Tab keyed following the last 'word' on the bottom or only line of the field.
7	Query keyed in position one of any field line.
11-24	Program function 1-14 keyed at any point of the field.

A code of 254 is returned by PANELF if the function code number is not one of those listed above. In addition the display and 'unshow' routines return codes as follows:

255	Field to display overlaps screen edge
254	No memory available to save screen contents under overlay field.
254	No saved data found to redisplay on 'unshow'
253	No more **PAFSVA** pointers free to save screen contents under overlay field.
253	Field to 'unshow' has no valid **PAFSVA** pointer number.

When user validation subroutines are called during data entry to a field it is sometimes necessary to display another field, perhaps an error message or additional instructions. This will almost always involve a 'recursive' call to **PANELF**.

For simplicity, and to save space, **PANELF** is not designed to be truly recursive: for example it makes use of a static area **PACRF** to hold the current field control block, which is also made available to the validation routines.

PANELF

The C language link module does, however, permit the **PANELF** subroutine to be called, for display only, from within a key, character, or line validation subroutine, by saving and restoring all the static information used by **PANELF** when it is called in this way. Note that no check is made that **PANELF** is called for display only. If **PANELF** is called recursively to read or validate a second field during validation of the first, this will result in 'nested' recursion, and the original details will not be correctly restored.

SEE ALSO

PACVAL, PAKVAL, PALVAL, PASCSE

NAME

PANELS - perform panel screen function

USAGE

```
#include <pnl5.h>
int ret, func;
ret = PANELS(func);
```

DESCRIPTION

The **PANELS** function is the single entry point for all of the panel screen-level functions. It performs the function **func** on each field in **PAFPT**. Names for the numeric values if **func** are defined in **pnl5.h** and are also used below.

PANELS processes fields using **PAFLX** as an index for the **PAFPT** field pointer array, under control of **PASET**. **PAFPT** will normally have been set up by **PAUSEP**. **PANELS** processes fields at all 'overlay levels' lying between **PAPSLO** and **PAPSHI** inclusive.

(Code 0) No action results for function code zero.

(Codes 1 - 40) Perform the operation defined by the corresponding **PANELF** function code on all fields. *WARNING:* the selected function may not be appropriate for all fields: for example, initialised output-only fields would be cleared by function 32.

For **PANELS(funsh)** the fields are processed in reverse sequence, so that higher numbered levels are lifted from the screen first. This is not foolproof, but it is more likely to be right than leaving high numbered levels to the end.

pdisp (Code 101) Display all the panel output fields. This function would normally be preceded by a call to **PAFUNC(3)** to clear the screen.

pshow (Code 102)	Display the panel 'input' fields.
pzero (Code 103)	Clear all panel input fields to their fill character. In a 'mask' field, only entry positions are changed.
ppadf (Code 104)	Pad all input fields on the right with fill.
prjnf (Code 105)	Performs the 'line operation' on each input field. The effect of this function depends on the validation code for line validation, but normally will right justify and format any numeric fields.
pdelf (Code 106)	Delete any fill characters from input fields, and replace them with blanks.
pread (Code 107)	Read the input fields for the panel. The function will evaluate the return code for each field and will move between fields according to the direction implied by the function which caused the field input to be terminated. The general linear motion is defined by the sequence in which the panel was written by **panel**. An escape of PF key will cause an immediate return, with the return value as defined for **PANELF**, and with **PAFLXS** set to the field number from which exit occurred.

The **PANELS** return value is undefined except for the **PREAD** function.

SEE ALSO

PAFLX, PANELF, PAXVAL

NAME

aneltc - extract data from termcap

USAGE

paneltc [*termname*]

DESCRIPTION

The **paneltc** utility program is provided only in environments supporting a 'termcap' (terminal capability) file. It reads the entry for the current **TERM** environment value (or for *termname*, if specified), and creates a **PANEL.VDU** file using the data found. Any existing **PANEL.VDU** file is overwritten.

Error number 13 indicates that no matching termcap entry could be found: any other error indicates that the data could not be onverted into the panel format. **paneltc** processes the basic screen output functions, the screen size and cursor positioning, 'standout' and 'underline' attributes, and 'box' graphics characters. No attempt is made to process codes generated by the keyboard, and a standard set of keyfunctions is defined.

After running **paneltc** you should probably use **tailor** to add additional highlighting functions if available, and to define your own choice of function keys.

SEE ALSO

PANEL.VDU

PANEL.VDU

NAME

PANEL.VDU - screen and keyboard information

DESCRIPTION

The **PANEL.VDU** information has seven sections as follows:

Identification
Screen Functions
Visual Attributes
Screen Size and Cursor Position
Delay Parameters
Keyboard Functions
Comments

A formal record layout of the record appears at the end of this section. Note that this record layout is provided for documentation purposes only, and that the PANEL run time subroutines are not designed to provide access to the record itself. Some environments supported by the PANEL system will not use the **PANEL.VDU** file to interface screens and keyboards.

Three sections of the record, those for VDU Functions, Attribute Definition, and Cursor Addressing, contain control sequences sent to the VDU to perform various functions. In each case the actual control sequences are held in a data area for each section, and the sequences for each function are defined by means of a length and address in the data area. This allows the maximum flexibility in the lengths of the various control sequences: the PANEL interface does not specify a maximum length for individual control sequences, only for the total of each section. In cases when identical control sequences are required (e.g. 'attribute off' for some screens) space is only taken up for a single definition in the data area, with each function automatically defined as the same length and address.

In the record data areas, any characters other than 0xff are treated as data characters and transmitted to the terminal. 0xff is normally a signal for a delay sequence to be initiated, and is also

used to signal special functions in the cursor addressing and 'attribute on' sequences. These features are described below.

The 'comment' areas of the record have a separate section for each part of the record. It is particularly important for the comment fields describing the keyboard functions to be set up correctly, because **keyboard** can be used to display the key assignments described in these fields to an operator. The first comment field can also be displayed by the tailor program to assist in selecting a parameter set, and should contain a fuller description of the configuration than is allowed by the ten-character terminal name.

Terminal Identification Section

A ten-character field is provided to identify the system or terminal to which the parameters apply. This field is displayed by some of the programs in the PANEL system and is also made available for access by user programs if required.

It is strongly recommended that any user-written program which uses terminal dependent features other than through the PANEL interface should check this identification field first. This technique will allow the application to be run on some other terminal if required in an emergency.

Screen Functions Section

Up to ten screen functions may be defined, of which six are used in the PANEL system. The other four are available for special purposes in user-written programs. A data area of 128 characters is available for specifying the control sequences.

The details of the use of the functions are as follows:

INIT (VDU Initialisation)

This control sequence should be sent to the terminal at the start of every program. It is normally used for some or all of the following functions:

- Ensuring that the VDU is operating in the correct mode.

- Selecting the correct character set
- Setting the display mode for the cursor
- Setting default video attributes or background colour
- Programming any programmable function keys
- Setting the screen width or character size

TERM (VDU Termination)

This sequence is used to deprogram any terminal settings made in the initialisation process which are not the standard ones for the installation. As an example one might wish to reset the screen width to its usual value, say 80 columns, after an application which required a 132 column screen.

The PANEL system programs will attempt to send this sequence before terminating, although it may not be possible in the case of certain abnormal program exit conditions. It is the responsibility of a user program to call for this function sequence to be transmitted before terminating.

It is recommended that the termination sequence should not include codes which cause the screen to be cleared. In many programs a fatal error is signalled by an error message prior to termination, and this will not be visible to the end-user if the subsequent termination sequence clears the screen.

CLS (Clear Screen)

This sequence should clear the screen and leave the cursor in the top left hand corner. On a terminal which uses attribute bytes for highlighting, it is important that they are all cleared by this code.

EEOL (Erase to end of line)

This sequence should preferably clear the line from the current cursor position, although it is acceptable for the whole line of the cursor to be cleared since the PANEL system only uses this function after positioning the cursor at the left side of the screen. If this function is not defined (ie length of the sequence is zero) then the PANEL system will output a suitable number of space characters to clear the line.

BSC (Backspace Cursor)

This code sequence should be defined as the most direct sequence which will move the cursor left one position. The PANEL system will never call this function when the cursor is in the leftmost column of the screen, so its effect there is immaterial.

Note that there need be no connexion between the code defined for this function and that generated by the key which is selected to indicate that the cursor is to be moved left.

BELL (Bell or screen flash)

This function will be defined on most terminals as 0x07, which will cause the bell or beeper to sound. If, exceptionally, the terminal has no bell or beeper, it is sometimes possible to specify that the whole screen should flash by programming for 'reverse video', 'delay', and then 'normal video'. The PANEL system normally uses this function to indicate either an error or that operator intervention is required.

SPR1 - SPR4 (Spare Functions)

The spare function positions are provided to allow user-written programs to make use of screen features other than those used by the PANEL system. They can be used in the development of applications which are to be used on a range of terminals which have different codes to perform the same functions.

Delay Parameters Section

There are two delay parameters which can be specified to the PANEL system, each by means of a single-byte value from 0-255.

DELA (1) (Screen Output Delay)

This value controls the amount of delay introduced by the 0xff character in output function sequences. The delay method is implementation-dependent. As a guide, a value of 10 in the screen output delay should cause a delay of about one character. This means that a value of 100 would introduce a delay of about ten

characters for each 'delay' character in the output string.

DELA (2) (Keyboard Input Delay)

This value controls the wait time which is introduced after detecting the first character of a multi-code key sequence. Its principal use is to allow the use of a single 'escape' key on keyboards which generate codes prefixed by the escape character for other functions such as 'cursor arrows'. After detecting the prefix character, the program waits for a time specified by this parameter. If a code is detected from the keyboard during this period, then it is assumed to be part of a multi-key sequence and tested as such to determine if it is a valid sequence; if not, the prefix character is assumed to be a single code, and is tested to determine if that code is defined as a keyboard function.

The correct value should be determined by experiment in systems supporting this feature. As a guide, a value of 20 is recommended for a 5Mhz 8086 operating at baud rates from 1200 to 9600, but a value of 4 or less may be sufficient for a system with integrated screen and keyboard. Too large a value will cause a noticeable sluggishness in responding to a single keyed prefix character, and will result in the system 'beeping' on fast keying as a result of multiple keystrokes being taken as part of one sequence. Too small a value will result in cursor movement or other function commands being interpreted as the single defined prefix. These problems can only affect keyboards which generate multiple key sequences.

If this value is set at zero, a different method of detecting and identifying input sequences is used. Characters are analysed as they come in, and first examined to see if a defined prefix is present. If so, characters are read until a sequence is either recognized or rejected. This method is the recommended one for use if the system codes allow it, but it does mean that a single 'escape' or other character which is also defined as a prefix cannot be used as a 'single' function key because it always causes a second 'read' from the console.

If no keyboard status is available from the operating system the PANEL subroutines will force the second delay parameter to zero even if another value has been entered.

Visual Attribute Section

This section of the VDU information file allows a wide variety of highlighting methods to be specified. Up to sixteen different highlight types or colours can be specified, and there are currently four different 'highlight method' codes. The code sequences for all the on and off sequences must fit within the space of 128 bytes.

It is recommended that highlight types 1 and 2 are reserved for simple screen functions such as inverse video, which are available on many terminals. It helps transportability if the more exotic functions are kept to the higher numbered highlight types.

METH (0-15) (Highlight Method)

It is necessary to specify a *highlighting method* for each of up to sixteen *highlight types* which are available. Method code zero indicates that the corresponding highlight type is not defined. The other codes have meanings as follows:

1 Attribute bytes are needed which take up a position on the screen preceding and following the data field. The visual attributes of characters already on the screen between the attribute bytes are changed when the attribute sequences are written.

2 Attribute bytes are needed which can be overlaid by data characters and which take effect from the character position following the attribute byte.

3 Attribute bytes are needed which can be overlaid by data characters and which take effect from the character position of the attribute byte.

4 A control sequence is sent to the screen to turn 'on' a particular set of attributes for characters subsequently transmitted to the screen, followed by a control sequence which turns 'off' that set of attributes and restores the default attributes. No screen positions are allocated for the control sequences.

8 Special code for border definition in which the 'on' sequence defines the default border and the 'off' sequence defines an alternate border. For more information, see **borders**.

In conjunction with any of the above codes (normally with method 4) a set of two-byte codes may be placed in the 'on' sequence, each of which starts with the character 0xff. When followed by a zero value, this code introduces a delay. A following code of 1 to 4 corresponds to the code letters P, N, S, and M respectively.

The 'method' selected for each highlight type controls the use of the 'on' and 'off' sequences which are described below.

For the **panel** screen editor program, it is recommended that one highlight type (e.g. inverse video) is specified for use by PANEL to highlight messages and to indicate 'selected' fields. This is done by setting the high bit of the method code for *one* of the highlight types (e.g. enter 132 instead of 4).

Currently, **panel** only supports the use of methods 3 and 4 for this purpose. This message line highlight specification will also be used by default in developed application programs.

Highlight method 1 can also be used to output single-byte visible characters instead of control sequences. For example, the 'on' sequence can be set to '[' and the 'off' to ']' which will cause fields to be delimited with these characters.

ON0 - ONF, OFF0 - OFFF (Highlight On and Off)

The sixteen highlight sequences are numbered 0 to F, using hexadecimal notation. In general, the 'on' sequence precedes the field and the 'off' sequence follows the field.

For methods one to three, the PANEL system writes the 'off' attribute byte first, following the field, and then the 'on' attribute preceding the field. This technique helps to avoid the remainder of the screen 'flashing' as would be the case if the 'on' attribute was sent first.

Normally, highlight type zero, the default, is set to method zero and no highlight sequences are specified for it. However if all the attribute methods are code 3 or 4 it may possible to dispense with all the 'off' sequences, and use the 'on' sequence for type 0 to reset any attributes to the 'normal' state. In this case the method code for highlight code 0 should also be set to 4.

Cursor and Screen Size Section

The PANEL system will send up to four different sequences to the screen for addressing purposes. Firstly, different sequences are provided for 'visible' and 'invisible' cursor definition. *Both* sequences must be specified, even if they are identical. Some terminals allow direct screen buffer addressing, so that characters can be written to the screen while the cursor is positioned elsewhere. If this function is available, the PANEL system will use it for all writing to the screen, only moving the cursor to fields where entry is required. This provides a much more restful life for data entry operators, since the cursor is not constantly flashing bout the screen. On systems without this feature, it may be simulated if the cursor display is programmable, by including a sequence to turn off the cursor display in the 'screen buffer addressing' sequence, and one to turn it on in the 'cursor addressing' sequence.

Secondly, the PANEL system allows for different control sequences to be used for two different halves of the screen. There are terminals which have different row bias values for the top and bottom halves of the screen, and in some 132-character line terminals the cursor positioning lead-in sequence, and the bias, are different beyond a particular column position. The PANEL system only allows one division of this type, which may be 'vertical' or 'horizontal'.

The convention used for row and column numbers in the PANEL system is that the top left corner of the screen is (0,0), and all bias values are calculated on this basis.

Two different methods of specifying position values are in common use, either a binary value in a single character, or ASCII coded decimal values taking one, two, or three character positions. The

PANEL.VDU

PANEL system can output sequences using either method, and allows an ASCII sequence to be defined either as variable length or as a fixed length of two or three characters.

The control sequences for this section must fit into a data area of 64 bytes. As before, each character other than 0xff is treated as a normal data character, but the 0xff character is in this case treated as the first of a two- or three-character sequence. If the 0xff is followed by a binary zero value, it signifies that a delay is to be introduced as described above. The other valid characters following the 0xff character are codes of binary one through binary eight. In each case the next following character is treated as a bias value. The meanings of the codes are as follows:

1 Binary column coordinate
2 Decimal column coordinate (V)
3 Binary row coordinate
4 Decimal row coordinate (V)
5 Decimal column coordinate (2)
6 Decimal row coordinate (2)
7 Decimal column coordinate (3)
8 Decimal row coordinate (3)

As an example, the positioning sequence for an ADM3a terminal (escape = y x) would appear as:

27 61 255 3 32 255 1 32

A total of 64 bytes is available for the positioning sequences, and the detailed specification of the parameters in this section is as follows:

MAXC, MAXR (Maximum Column and Row Numbers)

These values specify the size of the screen and normal values for the standard 80 by 24 screen are 79 and 23. (Remember that the to left corner is 0,0). The values are held in single bytes and the maximum is therefore 255. The practical maximum is the space required by the **panel** editor to edit the screen layout, as well as the availability of CRT displays.

XDIV, YDIV (Vertical and Horizontal Divisions)

These two values specify the screen coordinate after which the second set of cursor addressing sequences will be used. Only one of these values may be used, and the other should contain a value larger than the maximum column and row numbers (typically 254). In the majority of cases the screen is addressed without any discontinuity, and both values are then set to 254.

When the cursor addressing or screen buffer addressing function is invoked, it will use the appropriate second sequence if either the 'x' coordinate is greater than XDIV or the 'y' coordinate is greater than YDIV.

SBA1, SBA2 (Screen Buffer Addressing)

These sequences specify the codes for direct screen buffer addressing for the first and second halves of the screen. If the screen buffer addressing function is not available, the sequences hould be the same as those for cursor addressing. The second sequence is not defined if the entire screen is addressed in the same way.

CUR1, CUR2 (Cursor Addressing)

These sequences specify the codes for cursor addressing. The second sequence is not defined if the entire screen is addressed in the same way.

Keyboard Code Section

All function keys used by the PANEL system are user-definable. The system can recognize function keys which generate single codes or code sequences of any length. Multiple code sequences are identified using the following recognition strategy:

Up to four different single characters may be defined as a 'prefix' code or 'lead-in' for multiple sequences. When a prefix character is read from the keyboard the system first waits for a specified period to see if any codes immediately follow. This period is determined by the input delay parameter. If no character is

detected, the first input character is tested against the function key list to see if it is defined as a single-code function. This will often be the case with the escape key.

If a character follows within the time limit, the PANEL input routine enters a 'multiple code' mode. First, the system can be set to ignore a specific number of characters following each prefix. This allows for keyboards where the keys generate a lead-in sequence of more than one character. Then the significant character of the sequence is read and the character pair of prefix and significant character is tested against the function key list. The special case of two escape characters is detected even if a code is always to be ignored after a leading escape.

If a match is found in the list, PANEL can then be instructed to ignore a further number of characters, which may be different for each function. This allows for keyboards which append trailing characters after certain function key sequences. If the sequence is not identified as a valid function any such trailing characters will either be treated as data or as the start of a new input sequence.

The only restriction on multiple code sequences is therefore that for each prefix, there must be a fixed position in its possible following characters at which a unique identifying code appears.

The Keyboard Code section of the record defines the prefixes, the numbers of ignored characters, and the table of two-byte entries which holds the significant characters for each function. The second byte of single code function entries is null.

PREF (Prefix Codes)

Up to four different codes may be defined as prefixes. Any character which appears in this table will be treated as a prefix character as described above. Empty entries cause 'null' to be treated as a prefix: this feature implements the standard prefix for the IBM Personal Computer.

IGNP (Prefix Ignore Numbers)

Four numeric values, each corresponding to a prefix, indicate the number of immediately following characters which are to be ignored after detecting a prefix.

Input Function Table

This table of twenty two-byte entries contains the character-pairs which will be recognized as the following functions:

1	CR	Carriage Return
2	ESC	Escape
3	BS	Destructive Backspace
4	LEFT	Cursor Left
5	RT	Cursor Right
6	UP	Cursor Up
7	DOWN	Cursor Down
8	ERAS	Erase Line
9	INS	Insert Character
10	DEL	Delete Character
11	LINS	Insert Line
12	LDEL	Delete Line
13	TAB	Tab
14	QRY	Query
15	HOME	Cursor Home
16	PF1	Program Function 1
17	PF2	Program Function 2
18	PF3	Program Function 3
19	PF4	Program Function 4
20	PF5	Program Function 5

The detailed reaction of the PANEL system or of a user-written program to each function will depend on how each program has been written.

IGN (Function Ignore numbers)

Twenty numeric values, each corresponding to a function, indicate the number of immediately following characters which are to be ignored after identifying a function.

Extended Program Function Key Table

For special applications, the number of 'program function keys' available can be increased above five. This can only be done for systems which generate a maximum of two codes for each key, that is, the IGN table is empty because no trailing keystrokes have to be ignored. The input function table can conveniently be extended into the IGN table: when this has been done, a terminator character of 0xff is placed at the end of the extended table. The normal maximum extension of the input table is therefore nine character-pairs, making a total of 14 definable 'program function keys'. The presence of the special terminator indicates that extra function keys have been defined, and its position defines the number of such keys. Further extension of the table can be achieved by modifying the code generated by **vdugenc** to increase the **CPAVDU** size, and setting **PAKTS** to a value higher than 20 to signify the number of bytes to be scanned for the terminator character.

Comments Section

A space of 512 bytes is provided for comments on the customising. This space is divided as follows:

80 bytes:	Screen output functions
32 bytes:	Delay specification
10 bytes each:	16 Attribute types
40 bytes:	Cursor positioning
10 bytes each:	20 Keyboard functions

IBM Personal Computer Tailoring Features

When tailoring for the IBM Personal Computer or compatible systems under PC-DOS or MS-DOS, the Terminal ID field must start with the five characters 'IBMPC' (in capital letters). This signals to PANEL routines that IBM-compatible ROM BIOS calls can be used. If the ID field starts with 'IBMPC' it is important that you do not attempt to run any PANEL system programs on a non-compatible system: PANEL functions will attempt to issue software interrupts to access IBM PC BIOS functions which may have unexpected consequences in another system.

The special functions described in the remainder of this subsection for the IBM Personal Computer are brought into operation by the 'IBMPC' at the start of the terminal ID field. When using the IBM Personal Computer with a terminal driver which accepts escape sequences, these initial five letters of the terminal ID code must not be used. This applies to the PC-DOS 2.0 ANSI terminal driver as well as to CP/M-86.

Screen Functions Section

For the IBM Personal Computer, the only sequences which should be coded are those for the INIT, TERM, and BELL sequences. Any other function sequences will be ignored.

The BELL sequence can be left empty if you wish to silence the beeper, or have the single value 7 to sound the standard 'beep'. If two values are defined, the first is a frequency divisor and the second is a duration. Increasing the first value lowers the frequency.

The INIT and TERM sequences are set up in a special way: the first byte signifies the colour screen mode as follows:

0	40 x 25 monochrome
1	40 x 25 colour
2	80 x 25 monochrome
3	80 x 25 colour
7	monochrome monitor

The second byte has the 'base' screen attribute, to which the screen will be set by PANEL on a clear screen operation. A full description of the codes for this attribute will be found in the IBM Personal Computer Technical Reference Manual. In summary, the codes are as follows:

- Bit 7 set selects a blinking foreground character
- Bits 6, 5 and 4 set up the background colour
- Bit 3 selects high intensity monochrome or alternate colour
- Bits 2, 1 and 0 set up the foreground colour

The colour codes are defined as follows:

	Colour	Monochrome
0	Black	Non-Display
1	Blue	Underline
2	Green	
3	Cyan	
4	Red	
5	Magenta	
6	Yellow	
7	White	White

The third byte is an attribute value which selects the border colour for the display. Bits 2, 1 and 0 specify the border colour.

If fewer than three bytes are specified, the PANEL routines will omit the relevant functions. The initialisation sequence clears the IBM PC screen.

The termination sequence is coded in the same way as the initialisation sequence, and performs the same functions on exit from the program. Note that if you specify a termination sequence the screen will be cleared on program exit.

Visual Attributes Section

The method code for each attribute which you wish to use should be set up as 4, except for the attribute to be used for **panel** messages which should be set up as 132 (see above). The 'attribute on' sequence consists of a single byte, defined as described above for the second byte of the terminal initialisation sequence. The 'attribute off' sequence is not used.

The 'highlight modifier' is implemented for the IBM Personal Computer as follows: the actual value of the modifier is combined by 'logical or' with the attribute defined in the 'on' sequence for the highlight type. This allows selection of more combinations of background and foreground colour than are available with the PANEL's 16 highlight types. Remember that the 'modifier' only works when a method has been defined for the highlight type.

Cursor and Screen Size Section

The MAXC, MAXR, XDIV and YDIV parameters are set up as for a standard system. SBA1 and CUR1 each contain two bytes giving the start and stop line numbers for the cursor within a display cell, in the range 0..13 for monochrome and 0..7 for colour. For SBA1, these values should be coded so that the cursor does not appear. The values for CUR1 can be used to select a line or block cursor as required.

Note that if you use a higher value than 7 to set a monochrome cursor and then run the programs on a colour screen the cursor may completely disappear. The same effect occurs on a system such as the COMPAQ computer which combines mono and graphics on one display board.

You should ensure that the value of MAXC is set correctly to 39 or 79 to match your choice of colour mode.

TI Professional Computer Tailoring Features

The Texas Instruments Professional Computer has no 'background' colour facility, but a full set of attributes such as inverse and underline is available for each colour.

The provided screen attribute parameters for types 4, 5, and 6 are set up for normal, reverse video, and underline respectively, and require the colour to be defined by means of the highlight modifier according to the table below:

	Colour
0	Black
1	Red
2	Green
3	Yellow
4	Blue
5	Magenta
6	Cyan
7	White

PANEL.VDU

Wang Professional Computer Tailoring Features

The monochrome mode for this system is basically standard ANSI. The 'screen buffer address' mode is implemented in the supplied tailoring file by setting the 'hold cursor' mode, leaving the cursor static but switched on, during a write operation. This is generally preferable, when available, to switching off the cursor entirely, as it gives the operator a clearer view of entry positions on the screen.

In the colour mode on the Wang Professional Computer, the foreground and background colours are also selected separately from the display attributes such as inverse and underline. The default correspondence between numbers and colours can however be changed using an initialisation string as described in the Wang 'Program Development Guide'. This guide also gives details of how to select colour or monochrome on systems with more than one screen. It is suggested that the initialisation string for this system be set up with some care to correspond to your application requirements.

The Wang Keyboard is unusual in not having an 'escape' key. In the supplied PANEL tailoring file the 'backtab' key is used for the escape function because it is positioned on the keyboard where most people would look for an escape. The EXEC key is used for PANEL function 1 and the COPY key for PANEL function 5.

Victor 9000 / Sirius 1 Tailoring Features

The supplied files contain standard escape sequences for a 25 by 80 screen, and the initialisation sequence is used to set the correct operating modes. The clear screen sequence clears the bottom line of the screen separately.

The fact that all the keyboard codes are 'soft' means that there is no universal 'standard' keyboard layout. Use the **tailor** 'interactive keyboard specification' to set the PANEL function key definitions to the codes actually generated by your machine.

ACT Apricot Tailoring Features

The description of the supplied files for Victor 9000 and Sirius 1 also applies to the ACT Apricot. The INIT sequence can be used to label function keys on the microscreen, but no standard labels are provided. If you use the microscreen, the TERM sequence should reset it to the standard date and time display.

Hewlett Packard 150 Tailoring Features

The supplied file includes the standard initialisation sequence described in the HP Development Tools manual. The terminal name starting with the five characters 'HP150' causes **pantest** to switch between fields using the touch feature.

The source for **PANTESTC** and for validation functions show how touch fields can be automatically defined and supported. The supplied code requires special tailoring entries for SPR4, CLS, DELA(2), and PF4.

PANEL.VDU

The **PANEL.VDU** structure is defined as follows:

```
struct pavdu              /* VDU and Keyboard Info File Record */
 { uchar trmid[10];       /* Terminal or System ID */
   uchar addout[10];      /* Offset (base 1) in outdata */
   uchar lenout[10];      /* Length in outdata */
   uchar outdata[128];    /* Basic output sequences */
   uchar delay[2];        /* First o/p then i/p delay */
   uchar method[16];      /* Highlight method for types 0..15 */
   uchar addon[16];       /* Offset (base 1) in attrdata */
   uchar lenon[16];       /* Length in attrdata */
   uchar addoff[16];      /* Offset (base 1) in attrdata */
   uchar lenoff[16];      /* Length in attrdata */
   uchar attrdata[128];   /* Attribute ON and OFF sequences */
   uchar maxc;            /* Maximum column number (base 0) */
   uchar maxr;            /* Maximum row number (base 0) */
   uchar xdivide;         /* Vertical division point */
   uchar ydivide;         /* Horizontal division point */
   uchar addsba[2];       /* Offset (base 1) in cursdata */
   uchar lensba[2];       /* Length in cursdata */
   uchar addcurs[2];      /* Offset (base 1) in cursdata */
   uchar lencurs[2];      /* Length in cursdata */
   uchar cursdata[64];    /* Cursor and screen pos sequences */
   uchar prefix[4];       /* Keyboard prefix characters */
   uchar ignp[4];         /* Number to ignore after each prefix */
   uchar intab[40];       /* Table of keyboard pairs (up to 29) */
   uchar ign[20];         /* Number to ignore after each sequence */
   uchar commf[80];       /* Functions description */
   uchar commd[32];       /* Delays description */
   uchar comma[16][10];/* Attributes description */
   uchar comms[40];       /* Screen positioning description */
   uchar commk[20][10];/* Keyboard description */
   } ;
```

NAME

panfcb - field control block structure
panafcb - abbreviated field control block structure

USAGE

#include <pnl3.h>

DESCRIPTION

The structure to describe a field is named **panfcb** and is defined in **pnl3.h**. An alternative abbreviated structure omits certain fields and is named **panafcb**.

The fields in an fcb are as follows:

sx, sy, sh, and **sl** define the position and size of the field on the screen. If the field has a box/border it is not reflected in these dimensions and position.

sf is the defined fill character for the field.

The **so** field is used for two separate purposes in the field control block. Initially, it holds the 'overlay level number' for the field. The **panel** editor creates values of 0 to 9 in this field, but any values up to 127 are valid. When the field has been processed as an 'overlay' the **so** value has its high bit set and indicates the index of the pointer in the **PAFSVA** array which addresses the buffer containing the saved screen contents from 'under' the overlaid field. This **so** value remains until the next **PANELF(funsh)** or **PANELF(flose)** call for the field.

sa is a bit field defining the field entry attributes from the following list:

Upper case conversion	Initialised field
Echo input	Input allowed
Carriage return required	Key field

Wipe field on first input	Mask field
Right justify	Extended attributes
Numeric integer	Abbreviated format
Numeric currency	Border required
User defined	

Portable masks for the bits in the **sa** field are defined in **pnl3.h**.

Bits in the **sm** field define the allowable entry positions in a mask entry field. A portable macro is provided in **pnl3.h** to test these bits.

hightype and **highmodi** define the highlight type and modifier for the field.

dataheight and **datawidth** are the dimensions of the data area for the field. This may be larger than the size of the displayed field. **datablock** is a pointer to the start of the data area for the field. Note that the data lines may or may not be standard null-terminated C strings.

charvaln and **linevaln** are the numbers of the defined character and line validation subroutines for the field.

xa is a pointer to the extended attribute area for the field, or a null pointer if no extended attributes have been defined.

The remaining fields do not appear in abbreviated field control blocks:

fi and **fj** are hold the current cursor position within the displayed field. **di** and **dj** hold the current position of the displayed field within the field's data area. They contain the coordinates in the data area of the top left position of the field.

fname is the name of the field. Note that this is not a null terminated string.

Pointers to the **panfcb** and **panafcb** structures may be used interchangeably as arguments to panel functions, which determine automatically from **sa** whether or not the format is abbreviated. Note that some compilers may require casts to be used for pointer assignments if the same structure name has not been used throughout.

NOTES

The actual size of each of the structures will vary between different PANEL implementations. Take care to use the 'sizeof' function when allocating space for field control blocks.

pangenc

NAME

pangenc - generate C code to describe a panel

USAGE

pangenc [*-options*] [*panelname*]

DESCRIPTION

pangenc generates C structures to define the fields in a panel. Two files are normally created, an 'H' file to be included in your application to give access to named field data areas and field control blocks, and a 'C' file which is compiled to define and initialise all the structures and data areas required by the panel.

The output files from **pangenc** are named *panelname*.c and *panelname*.h. Existing files are overwritten. The code generation process is controlled by the program options, which may be specified at three stages: as defaults on line 9 of **pangenc.msg**, on the **pangenc** command line, and when the program prompts for a panel name in verbose mode. Options are all 'toggles', and specifying an option an even number of times is therefore a waste of effort. Line 10 of **pangenc.msg** defines (in upper-case letters) the option letters which are accepted at stages 2 and 3: this feature can be used to restrict operator entry of options to those appropriate to the environment.

Words on line 9 of **pangenc.msg** not starting with a hyphen are ignored and may be used as comments. Options may follow a hyphen in sets or individually. Case of options is not significant. **pangenc** processes only the first 15 tokens (filename or option set) encountered at each stage. Only one file may be processed by **pangenc** at each invocation.

Many of the option descriptions below will only be of interest to experienced C programmers who wish to port generated code between different compilers. The version of **pangenc.msg** supplied for each environment will define appropriate options for that environment.

A This option controls the initialisaton of character-string pointers. The default setting generates code with ampersands:

&SNAME[0], &MULTILINE[0][0],

and the modified setting generates:

SNAME, MULTILINE.

B The B option controls the generation of nested braces for initialisers in the panel control block. The default is to use nested braces as in K&R 8.6. The modified setting omits the inner braces.

C The C option controls the generation of nested braces for initialisers for multiline fields. The default is to use nested braces as in K&R 8.6. The modified setting omits the inner braces.

D This option radically changes the layout of the generated code. The 'H' file contains full definitions and initialisation for all fields, with the initialisation conditionally excluded when the module is included in your application program. The C file sets a define to force compilation of the initialisers, and then simply includes the 'H' file. The advantage of this option, which was the default in previous versions of PANEL, is that all the generated code appears in one place.

E Generates an *enum* set in the '.H' file to cross reference assigned field names to field numbers. No entry appears unless a field name other than the default name appears. This code can be extracted from the header file and used in your program which dynamically loades the screen layout.

F Option F causes 'full' field control blocks to be generated. The default is 'abbreviated' field control blocks, which omit the field name and certain other fields.

G Option G is for use with Computer Innovations C86 which objects to a cast on a constant initialiser.

H Option H avoids nesting include files when using the D option.

I This option generates quotes instead of angle brackets on the include statement generated for the 'H' file with the D option.

K Characters in string initialisers which have their high bit set are generated in escaped octal format.

L Character string initialisers are expanded as required to match exactly the declared string length. Required for QNX.

N This option is different from the others in that the it uses the letters following the option letter, up to the next white space, to replace the 'F' (if one) or the 'F' and the following underscore (if two) for automatically generated field names. This avoids duplicate field names between panels when using the D option. The -N option and following letters may also be placed in the panel comment field.

With the default setting of the D option the N option is unnecessary, since any fields not given an explicit name ar defined as static.

O This option selects generation of the old fcb format, excluding extended attributes, for use with versions of PANEL earlier than 5.30. Any extended attributes present are ignored.

P The P option selects quotes instead of angle brackets for the include statements generated for PANEL header files.

Q The Q option was provided for early versions of the QNX compiler, which required structure initialisers to be specified as if the structure was an array of ints.

S This option changes the byte sequence of the initialisers for the **sa** and **sm** fields in each fcb. Because **.pnl** files are portable, the bits in these fields have different significance for each processor type. For example, you should use the -s option when you generate code on an 8086 which is to be compiled on a 68000.

V The verbose option determines how much **pangenc** will tell you about its operation. The default setting requires **PANEL.VDU** information to be available and puts up a full

screen display, first to request a panel name if this does not appear on the command line, and then to display dots as each line is generated. With the modified setting (standard for UNIX) only error messages are displayed and **PANEL.VDU** information is not required.

X The X option causes an executable function to be generated in the 'C' file, which can be called to force loading of the module in an overlay environment.

If you wish to experiment with the above options, remember to check **pangenc.msg** to see what options are already in effect (line 9) and which are allowed to be entered (line 10). *WARNING:* some compiler/option combinations are spectacularly incompatible.

panpcb

NAME

panpcb - panel control block structure

USAGE

#include <pnl5.h>

DESCRIPTION

The structure to describe a panel is named **panpcb**.

The fields in a panel control block are as follows:

ml is the defined message line for the panel.

nf is the number of fields defined for the panel.

mf is the maximum field number used for generated field names.

dt and **dn** are the size of the data areas required for the complete panel and its 'input' fields respectively. The counts indicate raw data and do not include the nulls added to create C strings.

da is a pointer to the start of the data area used for a dynamically loaded panel.

pn is used to specify the panel name to **PALOAD**. A file type of '.pnl' is always assumed.

pc contains the comment or note field for the panel.

fa is a table of pointers to the field control blocks of the fields comprising the panel.

Each panel control block generated by **pangenc** has its own structure definition of the same format as the generic **panpcb** structure.

NAME

pantest - test and exercise panel layout
pantesti - test and exercise panel layout (memory-mapped version)

USAGE

pantest [*panelname*]
pantest *-m multfile index*

DESCRIPTION

The **pantest** program allows you to exercise a panel design for both testing and documentation purposes. It reads in a panel file using the dynamic load feature, displays it, and allows entries in each input field. This allows you to check if your attributes and validation are correct.

pantest is supplied in source form (**pantestc.c**) so that it can be compiled and linked with special validation routines.

The program can process multiple panels in one invocation, either in separate files or in a multiple file with index as created by **pnlindx**. After processing the fields defined at level zero you can display and data for each other level in turn by keying the level number at the main prompt. The screen display can be printed, or written to a text file to assist with application documentation. The text file is named **pantest.txt** and any existing file will have data appended.

SEE ALSO

PANTESTC.C source, **pnlindx**

PAOSINIT

NAME

PAOSINIT - initialise operating system
PAOSTERM - restore operating system defaults

USAGE

```
#ifdef OSINIT
PAOSINIT();
#endif
..
#ifdef OSTERM
PAOSTERM();
#endif
```

DESCRIPTION

The **PAOSINIT** and **PAOSTERM** functions are used when required by the operating system to set up the correct conditions for the program. In practice, they are required for UNIX operating systems and similar, and not required for DOS or CP/M.

The PANEL distribution package includes the source code for the **PNL1992C** module (99 indicates the library number) so that the functions can be customised to local requirements. As supplied, 'interrupt' and 'quit' signals are normally ignored, which may not be desirable during development.

NOTES

If a program including PANEL library routines crashes without completing **PAOSTERM** processing, the terminal may be left in a 'raw' state. Try '*stty sane ^J*' which sometimes makes things better.

PAPAGE

NAME

PAPAGE - select active and display page (PC-DOS only)

USAGE

```
int activedisp, working;
PAPAGE(activedisp,working);
```

DESCRIPTION

This function sets the 'active display page' and the 'working page' for use by PANEL functions. If the passed values for **activedisp** and **working** are different, the screen output functions will all operate 'invisibly' until a subsequent call changes the active display page. The calling program must ensure that the passed values do not exceed those supported by the hardware and operating mode in use.

SEE ALSO

PACHCL

NOTES

PAPAGE is only implemented in versions of PANEL which can use IBM Personal Computer ROM BIOS calls or memory-mapped I/O, and should only be used on a system including an IBM Color/Graphics Adapter or compatible equipment.

When the active display and working page numbers are different, use only **PACOUT** or higher-level PANEL routines. **PASOUT** and **PAROUT** use the ROM BIOS teletype emulation which is written on the assumption that the values are the same.

PASBAD

NAME

PASBAD - position screen writing position

USAGE

```
int x,y;
PASBAD(x,y);
```

DESCRIPTION

PASBAD is used to set the screen address for writing output to the screen. If supported by the screen, the cursor is turned off.

The column and row coordinates x and y are based on values of (0,0) in the top left corner of the screen. If either value is larger than the defined physical screen dimensions in **PAECB** the maximum value is substituted.

SEE ALSO

PAECB, PACURS.

NOTE

In environments in which PANEL buffers screen output, the **PASBAD** function flushes the buffer after sending the sequence.

NAME

PASCNL - return screen line contents

USAGE

```
#include <pnl5.h>
uint rownumber;
char p[133];
strcpy(p,PASCNL(rownumber));
```

The **PASCNL** function scans through all the fields currently defined in **PAFPT** to extract the data into a string. On the assumption that these fields have been displayed, the function returns a pointer to the contents of screen row **rownumber**.

If **rownumber** exceeds the maximum number defined in **PAECB** a line of hyphens is returned of the same length as the screen width.

Fields with an overlay level outside the range defined by **PAPSLO** and **PAPSHI** are ignored by **PASCNL**.

SEE ALSO

PAFPT, PAECB

PASCSE

NAME

PASCSE - Screen save enable
PALIFT - Lift a block of saved characters
PARDSP - Redisplay a block of saved characters

USAGE

```
extern int PASSEC;
int x,y,l,h;
char *p;
PASCSE(nel);
PALIFT(x,y,l,h,p);
PARDSP(x,y,l,h,p);
```

DESCRIPTION

The **PASCSE** function must be called before using any aspect of the PANEL 'windows' processing. Its function is to enable the saving of all characters written to the screen, so that a request to 'overlay' a screen field can first save the existing contents. The call to **PASCSE** should follow the **PAINIT** call; if **PAINIT** has not already been called, it will be called automatically with a null parameter.

The parameter **nel** controls the number of elements to be saved. A value of one saves only the data; a value of two saves data and highlight type; and three saves also the highlight modifiers. In normal use **nel** will be 3: only use 2 if you are sure you will never use highlight modifiers. The function allocates an area the size of the screen for each save element (data, type, modifier).

When IBM PC ROM BIOS functions are in use any non-zero **nel** value will be interpreted as 2, and the system will save the character and physical attribute. If **nel** is zero a special mode is se up in which displayed characters are not saved, but extracted whe needed directly from the display buffer using ROM BIOS calls. This technique is slightly slower but saves around 4k of heap space. With a non-zero value for **nel** only 'page 0' is saved.

PASCSE returns the number of elements actually set up to be saved, or -1 if insufficient memory is available (or if the automatically called **PAINIT** failed). The 'screen save element count' is also placed in **PASSEC**.

The **PALIFT** and **PARDSP** functions should not need to be called by an application program directly because the calls are made automatically by **PANELF**. The former lifts a region of the screen at coordinates **x**, **y** and of size **l** by **h** and places it row by row in the buffer at **p**. **p** had better be at least **l * h * PASSEC** in size.

PARDSP redisplays (with original highlighting, if **PASSEC** > 0) the saved buffer at **p** at the passed coordinates and size.

SEE ALSO

PAINIT, PANELF

NOTE

When 'screen save' is enabled the **PANELF** field highlighting routines ignore any highlighting whose method code requires attribute bytes, as these are not currently supported by the lift and redisplay functions.

PAROUT

NAME

PAEOUT - escape sequence output
PAROUT - character output
PASOUT - string output

USAGE

```
#include <pnl3.h>
char c, *p;
PAROUT(c);
PAEOUT(c);
PASOUT(p);
```

DESCRIPTION

PAROUT and **PAEOUT** use the lowest level provided by the operating system to write a character to the screen. If the character **c** is 255 then a delay is performed as specified by the first delay parameter in the **PANEL.VDU** information. All other characters are sent unchanged.

If the 'screen save' feature has been enabled, **PAROUT** will ensure that each character is saved with its highlighting parameters for use when a field is to be overlaid on existing data. **PAEOUT** is a version of **PAROUT** intended for sending control characters and escape sequences to the screen and no attempt is made to save the data with the 'screen save' feature.

PASOUT uses **PAROUT** to output a null-terminated string.

SEE ALSO

PANEL.VDU, PASCSE

NOTES

Some Panel environments buffer screen output. This will be the case when the symbol **FLUSH** is defined in **pnli.h**.

PAUSEP

NAME

PAUSEP - Use this panel in full-screen operations

USAGE

```
#include <pnl5.h>
panpcb pcb;
int ret, sfno;
ret = PAUSEP(&pcb, sfno);
```

DESCRIPTION

The **PAUSEP** function brings into use the specified panel **pcb** by moving its fcb pointers into **PAFPT**, the number of fields into **PANFL** and the message line into **PAMLN**.

sfno specifies a starting field number which will normally be zero, resulting in the field control blocks being loaded starting at field number one in **PAFPT**. When **sfno** is non-zero it is used as the field number in **PAFPT** at which this panel should start, and **PANFL** is increased if a higher numbered element is used than **PANFL** previously indicated. This feature allows a subsequent full-screen operation to operate on data from more than one panel. **PAMLN** is only set if **sfno** is non-zero.

SEE ALSO

PAFPT, PAMLN

PAXVAL

NAME

PAXVAL - general field exit validation

USAGE

```
int PAXVAL(func,fp)
int func;
panfcb *fp;
```

DESCRIPTION

PAXVAL is a routine provided by the application program which is called by **PANELS** for each field before and after processing. Source code for commonly required functions is provided.

The **func** value is the **PANELF** function in progress: it is positive when **PAXVAL** is called before the function and negative afterwards. **fp** is the field being processed. The return value from **PAXVAL** is not used.

The functions in the provided source file implement the display of the system date and certain support functions for other validation routines.

SEE ALSO

PANELS, Validation source

NAME

pd - PANEL data management

USAGE

pd [panelname]

DESCRIPTION

pd is the PANEL data file management program, which allows you to set up immediate data entry and record keeping functions without writing programs. Since the source is supplied, you can modify the program as required, although you will need a copy of C-TREE or a similar file access system to do this.

The program allows you to enter data according to your defined screen layout and validation specifications, and stores it in an ndexed data file from which you can retrieve it for viewing or printing. The data files are suitable for use directly by any program which can read fixed-length records, including a number of file-oriented word processing systems.

pd uses the C-TREE package from Faircom. C-TREE is a trademark of Faircom, whose address is:

Faircom (314) 445 6833
2606 Johnson Drive
Columbia MO 65203

pd can also be modified for use with your own file access method, and adaptations of **pd** may in future be provided for use with other data management packages.

The retrieval facilities allow look-up on any of up to ten key fields, which may be specified either precisely or generically by their leading characters. You can browse through the file in sequence based on any of these keys, and amend, delete, or print any record.

The data file layout is automatically implied by the 'input' fields defined on your screen. As supplied, only fields at level 0 are used by **pd**.

Preparing to use **pd**

If you create a screen panel design to be used with **pd**, there is only one additional requirement to be borne in mind when you are using **panel** to set up the layout: you must define at least one field with a 'K' attribute. Key fields must currently be single-line fields. Think carefully about where to put the key fields on the screen. The primary, or most important, key should go somewhere at the top or left, because it must be the first field when the fields have been sequenced by the **panel** output process.

All key fields allow duplicate keys, but in all but the first key field no entry is made in the index if the key field is empty.

pd *data files*

The first time that you run a new panel design, **pd** will create a data file (.dat) and index file (.idx) with the same name as the panel file. Either all or none of the files must exist. Files are assumed to be in the current directory unless a 'PD=' environment variable specifies another directory.

When the files are created, **pd** also creates a C-TREE ISAM parameter file (.p) which will be useful if you need to read the file in another C-TREE application.

The '.dat' file contains header information which takes a minimum of 128 bytes, but which is rounded up to a space equal to an integral number of records. It is therefore always possible to calculate where to start reading if you wish to read the file sequentially without using C-TREE. Note that deleted records in the data file start with a byte with value 0xff.

Using **pd**

pd has two processing modes, adding and retrieving records. The main prompt line is used to select them.

Adding new records

Just type them in. The **pd** screen entry features will take you from field to field, in the sequence you specified when writing out the panel definition. If you specified that fields needed a carriage return to complete, then you will need to enter one, and so on with all the field attributes. The standard validations are all available.

When you get to the end of the screen, **pd** will ask you whether you wish to file the record as is, change something, or abandon it. On the assumption that you may want to add several records, the system stays in 'New Record' mode until you press the escape function key at any time.

Retrieving records

In retrieval mode, you have to enter a key value into ONE of the key fields. Skip over them, if more than one, with the tab or carriage return key, and enter a value. The value entered may either be a complete key value, or you can enter a partial key terminated with that backwards oblique character (\).

As an example, in a file of countries, an entry of 'U\' would find 'UGANDA', 'URAGUAY', 'USA', 'USSR', and anything else you had entered beginning with 'U'. If the backwards oblique character is the first in the field, you will find all the records.

If no matching record is found, **pd** will tell you, but if there is a match, it will enter 'Scan mode'. First, the system does a quick scan through the index to see how many records match your key specification. Then it displays the first one, and allows you to browse through the file forwards and backwards with the F and B commands. You can even go outside the limits of your key specification, forwards or backwards up the file. Records are presented in the sequence of the key field being scanned.

There are a number of things that you can then do with the record displayed on the screen. You can print it, and it comes out on the system printer exactly as it is shown on the screen (less any highlighting). You can also amend or delete it.

pd

When you amend a record you can use any of the PANEL editing features which you used on entry. You are quite at liberty to change any field, including key fields, but if you change any key field, or delete the record, you will have to restart the scan as the record sequence may have changed.

You can quit retrieval mode at any time and return to the main menu.

NAME

pnlindx - create index for multiple panel file

USAGE

pnlindx [*-v*] *multfile index*

DESCRIPTION

pnlindx builds an index to a file containing multiple panel files. The data file is first built by concatenation: use a command such as:

```
copy  a*.pnl/b+b*.pnl  multiple      [MS-DOS, PC-DOS]
cat   a*.pnl  b*.pnl  >multiple      [UNIX, XENIX]
```

Both the above commands create a file named 'multiple' which contains all the panels with names beginning with 'a' or 'b'.

The **pnlindx** program creates *index* from *multfile* containing the panel names and starting record numbers. The *-v* option displays panel names as they are encountered.

Index files created by pnlindx are not portable between machines using different byte sequences for two-byte integers.

SEE ALSO

PAMULT, PNL files

NOTES

Panel description files which are to be indexed with **pnlindx** should only be renamed by rewriting with **panel,** since **pnlindx** uses the internal panel name to create the index. Do not use the operating system rename or mv command.

PNL files

NAME

PNL files - panel description files

DESCRIPTION

Panel description files consist of a set of fixed-length 32-byte records of similar format to panel control blocks and field control blocks. The first two records contain the panel header information and the panel comment field. These are followed by a set of records for each field.

The first record for each field contains the fcb information. This is followed by the initialisation data for initialised fields and then any extended attribute records for the field. It is always possible to predict the number of records to be read, so finding end of file indicates an error.

Fields are sorted on 'display direction' within 'input/output' (input first), within level number. Display direction uses the screen position to sort the fields so as to give a basically horizontal or vertical progression through the panel.

The first record of a panel file contains the panel name when written by **panel**, so panel files should not be renamed with operating system commands if this is of significance to you.

NOTE

panel creates backup files with extension **.pnb** before writing a file with a name which matches an existing file.

NAME

pnli.h - compiler selection header
pnlc.h - portability header
pnl3.h - field-level header
pnl4.h - validation header
pnl5.h - screen-level header

DESCRIPTION

pnli.h contains definitions for the compiler, hardware, and operating system in use. It controls the selection of definitions in **pnlc.h**. In some environments multiple **pnli.h** files will be provided to support different compiler options. These files will be identified as **pnli.h99** where '99' is the interface number, and should be renamed to **pnli.h** before use. In other situations only one file is necessary when, for example, the Lattice C compiler automatically defines internal symbols to indicate the compiler options in use.

pnlc.h is the PANEL portability header file. If you only plan to use one environment you can cut most of the statements out of this file if they are never going to be selected. The full file is provided for users who will be transporting PANEL applications.

pnl3.h and **pnl5.h** contain respectively the definitions required to interface with the PANEL subroutine library at the field level or the screen level.

pnl4.h is provided for use when compiling validation source files. It defines certain variables used for validation functions, and includes the other header files.

All the PANEL header files can be included in a program more than once without causing damage to anything other than your compilation speed.

tailor

NAME

tailor - screen and keyboard installation functions

USAGE

tailor

DESCRIPTION

tailor performs VDU and keyboard tailoring functions. It includes a great deal of explanatory material which will be displayed to help you make the necessary entries. This description is therefore designed merely to give a general idea of its capabilities, and to help you plan what you are going to do with it.

tailor can be operated without any external files other than **tailor.msg**, in which case all you can do with it is create **PANEL.VDU** which contains the details of the current active terminal. Normally, however, you will use **tailor** in conjunction with the **TAILOR.DAT** file which holds a number of sets of terminal definitions. You can then pick one of the predefined parameter sets and store it as **PANEL.VDU** to make it 'active', edit or modify it, and then save it again in **TAILOR.DAT** if you wish.

The most important aspect of the **tailor** operating environment is the screen and keyboard from which you run the **tailor** program. This need not be the target system for the current tailoring operation. If it is not, you will not be able to use the interactive keyboard specification option, but you can define all the keyboard codes by working out yourself how to recognize them.

The other thing you may not be able to do when tailoring for a different system or terminal is to use the facilities provided to test each output code. **tailor** does not prevent you from doing this, because you may know that the codes are the same on your terminal. Be aware, however, that another system's 'clear screen' code might well turn off the keyboard on your own.

You will find that **tailor** will ask you whether the basic screen

functions have been defined correctly for your terminal. If this is the case you will be spared the undignified sight of questions scrolling up from the bottom of the screen.

Using **tailor**

tailor can do three things: it can pick terminal details from those in **TAILOR.DAT**, it can save terminal details in **TAILOR.DAT**, and it can edit the terminal details. The 'active' terminal details are always held as **PANEL.VDU** and you will be asked whether you wish to update this file by 'saving' the changes after each editing operation. **tailor** determines which of these files are present and adapts its menus automatically so that you are not presented with impossible options.

Editing terminal details

The main editing menu allows you to choose the following things to edit:

- Terminal ID
- Delay parameters
- Basic Screen Output Functions
- Cursor Positioning and Screen Size
- Highlighting
- Keyboard Codes (Interactive)
- Keyboard Codes (for another terminal)

If you are starting from scratch it is a good plan to work down this menu from the top, because this way you will be able to test each function as you define it. Some of the functions require that the preceding items have been correctly specified.

For each section, there is space to enter a comment about what has been defined, which may be helpful if you cannot remember later what the codes were for. For the highlighting and keyboard codes there is a separate comment field for each element.

tailor *code input conventions*

tailor always displays existing codes as a sequence of decimal

numbers, but when it comes to entering a new sequence the only input convention is to enter the values any way you like. Just about the only reasonable entry method that **tailor** doesn't recognize is octal.

A code sequence is entered element by element, and since most sequences are fairly short, **tailor** requires that you enter the whole sequence even to change one already defined. If you make a mistake you can enter 'B' followed by carriage return to back up the list. For really bad mistakes, an 'A' code will abandon entry without changing the existing codes.

If you enter a numeric value, it is assumed to be decimal, unless it is followed immediately by an 'H' in which case it is treated as hexadecimal. You can also use any of the standard ASCII codes, if you can remember them, such as SOH and NAK and things like that. When the code to be entered is a letter (for example in a sequence such as 'escape - K'), then you can press the space bar to indicate 'use the following character' and then just type capital 'K'. Easier than looking up decimal 75 or hexadecimal 4B. Finally, 'control' characters can be entered by pressing the '^' key, and then typing the appropriate letter or symbol.

As an example, there are five different ways in which you can place an 'escape' character in an output sequence:

1. Enter '27' followed by carriage return
2. Enter '1BH' followed by carriage return
3. Enter 'ESC' followed by carriage return
4. Enter '^' followed by '['
5. Enter space, then press the escape key

In addition to all of the above, there are special alphabetic codes you can enter: for example a 'D' inserts a software delay into the sequence, of a length specified by the output delay parameter, and 'X' and 'Y', each followed by carriage return, can be used in a positioning sequence to indicate 'the coordinate value goes here and it's a binary value'. Details of the alphabetic codes are given below.

Interactive keyboard editing

Although it is fully explained in the **tailor** program, it is worth noting here that you specify your keyboard codes by pressing the actual key when asked to. **tailor** uses a full-screen display for this process and shows the code read as soon as it has been entered. When you have finished your entries and come to save the table, **tailor** works out the best way of recognizing each code. If it displays 'SINGLE' or 'MULTIPLE' in the status column, it has successfully determined its recognition strategy: if not, the offending codes will be marked as duplicates or as having no unique distinguishing features, and you will have to try again.

In rare cases, you will find that the status column contains a 'NOT VALID' message when the keyboard table is first displayed. This usually means that you have unsuccessfully defined the keyboard codes with the non-interactive option, perhaps by specifying a pair of characters of which the first has not been defined as a prefix. You can clear this error by respecifying the key. A 'null' entry (CTL-@) 'undefines' a function.

The interactive keyboard editing procedure does not currently permit more than five function keys to be defined. If you need to define more than this, and if the keyboard generates no more than two characters from any key depression, you may extend the 'keyboard recognition table' into the 'ignore table' by entering the character pairs generated by each function key using the 'K' option of **tailor** editing. The list of character pairs has to be terminated with a 255 value (which is displayed as 'D') to indicate that this feature has been used, and a maximum of nine extra function keys can be defined in this way. For further information, see the Tailor Interface specification details in the next section.

Currently the interactive keyboard editing feature requires a screen at least 80 by 24 to operate. If your screen is less than this size, you will have to use the non-interactive keyboard specification option (or buy a bigger screen).

Special Alphabetic Entry Codes

In any sequence a code of 'D' will specify that a delay is to be introduced at that point. The length of the time delay is specified by the first delay parameter.

For cursor addressing and direct screen buffer addressing the following alphabetic entries in a code sequence are used to indicate the position of a row or column number in a sequence. Immediately following the letter must be a value denoting the offset to be applied to the coordinate. As an example, the common positioning sequence of 'Escape Equals' followed by row and column coordinates offset by 20h is entered to **tailor** as a six-element sequence as follows:

ESC = Y 32 X 32

The full range of alphabetic codes for screen addressing is as follows:

X - Column value, single byte binary
Y - Row value, single byte binary
I - Column value, variable length ASCII
J - Row value, variable length ASCII
K - Column value, two-byte ASCII
L - Row value, two-byte ASCII
U - Column value, three-byte ASCII
V - Row value, three-byte ASCII

Another set of alphabetic codes are used in attribute 'on' sequences as follows:

P - Send the remainder of the sequence (max 8 characters) as a prefix for each character to be highlighted.

N - Use the next following byte as an increment for each character output (modulo 256). This feature can be used t convert alphabetic to 'control' characters for graphics purposes.

S - Set the high bit of each character output.

M - Use the 'highlight modifier' at this point in the sequence by adding its value to the next character of the sequence.

SEE ALSO

PANEL.VDU, TAILOR.DAT

TAILOR.DAT

NAME

TAILOR.DAT - tailoring data file

DESCRIPTION

The tailoring data file consists of an index block followed by a data block for each set of **PANEL.VDU** information. Each block is 1024 bytes long. The index block contains a 10-byte reference field, a single byte value of the number of data blocks, a filler byte, and up to 50 ten-byte index entries containing a copy of the **termid** field from the data block. The second half of the index block is unused.

Each data block is formatted as described under **PANEL.VDU**.

SEE ALSO

PANEL.VDU, TAILOR

NAME

alidation - standard validation functions

DESCRIPTION

The standard validation routines described below are those for which the source code is provided. If you have modified this code, the functions may not behave as described.

Character Validation 01

Instead of allowing all displayable characters to be entered, this routine restricts them to a user-specified subset. In the supplied programs, this subset is initialised to the letters 'A'..'Z', but it can be changed by using extended validation CL.

Character Validation 02

Useful for password fields, this routine displays any entered character as an '*', while saving the true entered value in the field. Note that characters are only translated on input: if you display the field unchanged the secret entry will be revealed.

Character Validation 03

Uses the 'picture' defined by extended attribute PV to specify validation separately for each character of the field.

Character Validation 04

Restricts the field entries to the digits '0'..'9', without validating the whole line as a 'number'. Useful for fields such as telephone numbers and dates which may need to be entered under mask.

Character Validation 05

Displays either 'Yes' or 'No' (provided the field is large enough) when either 'Y' or 'N' is entered. Other entries are rejected. If you have specified 'CR required' for the field, you will have the

opportunity to backspace to the start of the field and re-enter the value; if not, a single keyed valid character will complete the field. Function key usage is restricted, and you cannot leave the field without selecting one of the valid options. The emitted strings can be changed with extended attribute VL.

Character Validation 06

For those data entry operators who are used to a typewriter keyboard, this routine allows entered values of '<' and '>' to be converted to comma and full stop (period) respectively. These special characters are often found on the 'upper shift' position of those keys, and many punctuation errors can be avoided.

Character Validation 07

This validation routine implements a simple but effective word-wrap mechanism for data entry. It allows continuous keying into a multi line field, and if no space character falls at the end or the start of a line it will move an incomplete word from the end o one line to the start of the next.

The routine does not support dynamic reformatting of the field when editing already displayed data. If you have to edit the start of a line which has a completely filled immediate predecessor line, a space character (which will be thrown away) should be entered first to inhibit word-wrap.

Character Validation 08

This routine uses the associated VL extended attribute to step through a list of values when a single key is pressed. This routine is described in greater detail in **extended attributes**

Character Validation 09

This routine supports the vertical pop-up menu function **PAMENU**. It scans the first character of each line of the field to check for a match with the entered character. The entered character is ignored, but the match index is saved for use by **PAMENU**. Character validation 09 also causes **PANELF(freadu)** to suppress the display

cursor if possible. It is recommended that this validation number only be used from within **PAMENU**.

ine Validation 11

This routine validates a date entry in the format 'DD/MM/YY'. Any non-numeric delimiters are accepted. If the field size allows, the year may be entered with more than two digits and is checked modulo 100, i.e. the 'century' is ignored. If the date is not valid, the 'bell' is sounded, and the cursor returns to the start of the field. For this routine, there is no protection against an 'escape' from the field, perhaps after a partial incorrect entry. If the 'user' attribute (X) is applied to the field, this routine treats as valid a 'blank' field containing only fill characters in entry positions

Line Validation 12

Dates are checked in the same way as for function 11, but the input format caters for those countries where 'month' comes before 'day' 'MM/DD/YY').

Line Validation 13

Date validation as for 11 and 12 above, but format 'YY/MM/DD'.

Line Validation 211

Before display of the field, its contents are initialised to the content of the system date field, using the format 'DD/MM/YY'. This routine is of course only available when the operating system is able to supply the date.

Line Validation 212

As for 211, but date format 'MM-DD-YY'.

Line Validation 213

As for 211, but date format '19YY.MM.DD'.

Validation

SEE ALSO

attributes, extended attributes, PACVAL, PALVAL, validation source.

NAME

vdugenc - create source file from PANEL.VDU

USAGE

vdugenc *vduname*

DESCRIPTION

vdugenc is used to generate a C source file which can be compiled and linked with an application to 'hard-wire' it to a specific screen and keyboard. The output consists of a **PAVDU** array initialised with the contents of **PANEL.VDU** (from the current directory).

The *vduname* parameter must be supplied and has any existing filename suffix removed and replaced by '.c'. *vduname*.c will be overwritten without warning if it exists. **PANEL.VDU** is read from he standard place or extracted from **TAILOR.DAT** as described under **PAINIT**.

For systems which do not generate trailing characters after keyboard functions, it is possible to extend the keyboard table into the 'ignore table' by placing a 0xff flag in the last 20 bytes of the 512-byte **PAVDU** structure to indicate the end of the keyboard table. You can also extend the keyboard table even further by modifying the code generated by **vdugenc** to extend the array beyond 512 bytes as described in **PAINIT**.

SEE ALSO

PAINIT, PANEL.VDU

NOTES

In some environments it may be necessary to temporarily remove the empty object module pnl1993c (99 represents the relevant interface number) from the panel interface library or archive to avoid a 'multiply-defined symbol' message for **PAVDU**.

Vertical Scrolling

NAME

Vertical Scrolling of field contents

USAGE

```
extern int PASCRO;
PASCRO = 1;
```

DESCRIPTION

It is sometimes necessary to have a field data area which is larger than the displayed 'window' for the field, and to view different portions of the data. PANEL provides a built-in facility to scroll fields up and down automatically during a 'read' operation.

If the following three conditions are satisfied, fields will scroll vertically when the cursor is moved out of the top or the bottom of the field:

- Global integer **PASCRO** must be set to 1 (default is zero).
- The field control block must be 'full' not 'abbreviated', so that the additional fields holding the position of the display window in the data area can be maintained.
- The field data area height **dataheight** must be greater than the field displayed height **sh**.

Currently, fields cannot be generated automatically to satisfy the third condition. Instead the field control block or the data area must be set up or modified by the application program. The **testc.c** file includes an example of one method of doing this, but there are many other possibilities.

If your application requires the user merely to display the data by scrolling, without changing it, you should inhibit data entry for the field using the **PAKVAL** exit, just allowing the appropriate cursor movements to be made.

Vertical Scrolling

When the scroll conditions above are met, the line insert and delete functions will act on the whole data area instead of just the display window. Also, the home function has an extra stage which will move the display window to the top of the field data if the cursor is at the top left of the window but the window is not at the top of the data.

When a screen panel has several multi-line fields blocked horizontally across the screen you may wish to scroll all of them when one scrolls itself automatically. You can use the special exit to PALVAL (with a passed code value of -1) to redisplay other fields in such circumstances. If you do this, you may wish to inhibit line insert and delete for the field using PAKVAL.

Vertical scroll does not operate for the PAMENU function. You would need to change the suppled PAMENU source if this is a requirement.

Section III
PANEL for MS-FORTRAN

Using PANEL with MS-FORTRAN

Introduction

This section provides information about PANEL version 6.20 for Microsoft FORTRAN V3.3x under MS-DOS/PC-DOS.

If you have been using PANEL version 5.3/5.1 for Microsoft FORTRAN V3.2, some changes will be necessary to your screen panel designs and to your FORTRAN programs. We regret that it was not possible to retain upward compatibility and still give you all the new features of PANEL version 6, but we have attempted to keep the main principles of the interface the same. Version 6.20 of PANEL includes a wealth of new features which are now all accessible from FORTRAN. These include multiple overlaid fields and windows, pop-up help boxes, many extended validation functions, and a high-speed memory-mapped option for the IBM-PC and compatibles. In addition, you can mix C and FORTRAN validation functions within the PANEL library.

Currently, the PANEL FORTRAN interface is only available under MS-DOS and PC-DOS, but we do intend to release a version for Xenix.

Documentation

Since the PANEL FORTRAN interface now makes direct calls to a 'C' function library and can perform almost all of the functions that are included in our 'C' versions, most of the information you need to use the FORTRAN version will be found in the reference section of this manual, which primarily covers the 'C' versions.

The PANEL utilities such as the **tailor** program, the **panel** screen editor, and **pantest**, are common to all supported languages.

Using PANEL with MS-FORTRAN

The FORTRAN Library Implementation

The PANEL libraries, **PANEL04.LIB** and **PANEL04I.LIB**, contain mainly modules written in 'C', with a small amount of assembler code. Two of the modules, those for numeric validation and current date display, are currently written in FORTRAN.

Most of the principal PANEL functions are C routines called from FORTRAN by declaring an 'interface' defining the C parameters. PANEL global variables are not accessible directly from FORTRAN, so routines are provided to retrieve and set these values.

In accordance with our normal policy, all source code for validation functions is provided. This means that the FORTRAN version has a fair amount of C source code on the distribution disk. To modify the validation functions you can either recompile them with the Microsoft C Compiler Version 3, or replace or augment them with functions written in FORTRAN. The C compiler is not required in order to use PANEL for FORTRAN.

We use some of the utility routines in the *CEXEC* library provided on FORTRAN disk, so you will need to include this in your linkage command. Further information about linking appears below.

Changes from Version 5.3/5.1

This section gives guidance on modification of existing applications running with the earlier version of PANEL for FORTRAN. It cannot be an exhaustive list, as you will obviously have to examine all function calls to ensure compatibility.

- Panel description files (.PNL) now use the old 'spare' field as a level number. To ensure that these all start as zero, you should load each panel with the **panel** program using the '-z' option and save it again.

- The old FORTRAN interface used special line validation numbers to specify the number of decimal places of numeric amounts. This feature is now implemented as part of the *NF* extended attribute. You should therefore probably set up a *NF*

extended attribute for each numeric field, although standard integer and currency fields will default. The line validation number should now revert to zero for normal numeric fields.

- All generated code must be regenerated with **pangen04** V6.20.

- Each program must set up a dynamic allocation buffer and call the **PFALST** function before making any use of PANEL functions. The dynamic allocation buffer is used by PANEL for the screen save buffers, dynamically loaded panels, and temporary work areas used for extended attribute processing. If you use the 'I' library (which does not require a full screen save buffer) and only one or two screens, then 8kb may be sufficient. The maximum value is currently 64kb, but it is hoped to increase this in the a future release.

- If you wish to use help boxes and overlay fields you must call **PASCSE** shortly after calling **PAINIT** to enable the screen save function.

- Generated code for a panel now includes C-format field control blocks, which use pointers. The BLOCK DATA syntax does not permit pointer initialisation, so we fill these fields with offset values and require you to call a function **PFSETP** to convert these fields to genuine pointers before you use the panel (ie before the first use of **PAUSEP** for the panel). It is extremely important not to omit this call: before conversion to pointers the offsets refer to low memory addresses, and a call to certain panel functions without having converted the pointers may turn out to be the last thing your program does before you have to switch off the system. The **PFSETP** function does not need to be called for dynamically loaded panels.

- Because the field control blocks now contain real pointers, not the offsets into **PDATA** used in the old interface, we no longer need the **PDATA** parameter on any function call.

- Panel functions for which the 'C' syntax requires a pointer (eg **PANELF**) now require a pointer when called from FORTRAN also. This requires the use of the *LOC* function described in

Appendix A of the FORTRAN manual.

- Panel functions which take a string argument expect a 'C' format null-terminated string which must be set up in FORTRAN using the '[C]' attribute described in Appendix A of the FORTRAN manual. For consistency, we have defined these arguments as pointers also. An example of the syntax is therefore: `CALL PAMSGE(LOC('Hello'C))`.

- The numeric arrays **PANUMR** and **PANUMI** are used in much the same way as in the old version but we no longer provide named equivalences for the arrays for each panel. Instead the member of the array to be used is specified as the first substring of the NF extended attribute. The processing for numeric fields is described in the reference section. The **PANUMR** and **PANUMD** arrays no longer need to be passed to **PAUSEP** or **PALOAD**.

- The **PANFE** and **PANUNL** function to expand or unload numeric fields are no longer provided. Instead the **PANELF** *flnop* (code 35) or **PANELS** *prjnf* (code 105) function is used to expand values, and the read validation places numeric values in the array. This is consistent with the C language versions.

- PANEL global variables such as **PAFLX, PARDHI** etc, are no longer situated in a common block but have to be retrieved and set using the **PFGET** and **PFPUT** functions.

- The **PALOAD** function uses a C-format panel control block, into which you must place the filename at offset 47. See the **pantst04** program for an example.

Field Control Block format

The format of the field control block is shown using C syntax in the provided **pnl3.h** file. In FORTRAN, the following is a description of the contents:

Position	Size	Type	Contents
1	1	CHARACTER	X-coordinate of screen field
2	1	CHARACTER	Y-coordinate of screen field
3	1	CHARACTER	Length of screen field
4	1	CHARACTER	Height of screen field
5	1	CHARACTER	Fill character
6	1	CHARACTER	Level number
7	2	INTEGER*2	Attributes (bit field)
9	2	INTEGER*2	Entry Mask (bit field)
11	1	CHARACTER	Highlight Type
12	1	CHARACTER	Highlight Modifier
13	1	CHARACTER	Data area width
14	1	CHARACTER	Data area height
15	4	INTEGER*4	Data area pointer
19	1	CHARACTER	Character Validation Number
20	1	CHARACTER	Line Validation Number
21	4	INTEGER*4	Extended Attribute Pointer
25	1	CHARACTER	Cursor x-position in field
26	1	CHARACTER	Cursor y-position in field
27	1	CHARACTER	x-position of field home in data
28	1	CHARACTER	y-position of field home in data
29	8	CHARACTER*8	Field Name

The last five fields do not appear in abbreviated field control blocks. The extended attribute pointer addresses a 66-byte area, of which the first two bytes are a set of bits indicating which extended attributes are present, and the remainder is 16 pointers to the extended attribute strings.

Panel Control Block format

The format of the panel control block is shown using C syntax in the provided **pnl5.h** file. In FORTRAN, the following is a description of the contents:

Position	Size	Type	Contents
1	2	INTEGER*2	Message Line
3	2	INTEGER*2	Number of fields
5	2	INTEGER*2	Max generated field number
7	2	INTEGER*2	Total bytes in data area
9	2	INTEGER*2	Input field bytes in data area
11	4	INTEGER*4	Pointer to data area
15	32	CHARACTER*32	Comment of Note field
47	80	CHARACTER*80	File Name for Dynamic Load
127	n*4	INTEGER*4	Pointers to field control blocks

Direct Calls to C Functions

The following functions and subroutines are PANEL C library functions called directly from your FORTRAN program. Note that you should include **pnl3041f.inc** and **pnl3042f.inc** to specify the interfaces and variable types. In all the following function interfaces, the argument *xLOC* refers to a location pointer, the return value from the *LOC* function described in section A.6.3 of the MS-FORTRAN manual.

In general, the functions with names starting *PA* are calls to C library functions described in the main PANEL manual. Those starting with the letters *PF* are C library functions specific to the FORTRAN interface.

Examples of most of the function calls will be found in the provided **pantst04** and **test04** source files, and a full description of each standard function is in the main Panel Reference Manual.

The **PFALST** subroutine is used to set up PANEL's internal memory allocation functions. It passes the location and size of a buffer which must be defined in named common by your program. No check is made that the size matches the size of your common block. Do not attempt to modify this buffer during the execution of your program. The **PFALST** function must be called before any other panel routine.

```
INTERFACE TO SUBROUTINE PFALST [C] (BUFLOC,SIZE)
INTEGER*4 BUFLOC
INTEGER*4 SIZE
END
```

The **PAINIT** function initialises PANEL functions. It returns a zero value if **PANEL.VDU** information is available, or if the 'I' library has been linked (which does not use **PANEL.VDU**). The pointer references an override pathname for **PANEL.VDU**, normally LOC("C).

```
INTERFACE TO INTEGER*2 FUNCTION PAINIT [C] (PLOC)
INTEGER*4 PLOC
END
```

The PASCSE function enables screen save. It returns the number of elements to be saved, or -1 if insuffcient memory is available.

```
INTERFACE TO INTEGER*2 FUNCTION PASCSE [C] (NEL)
INTEGER*2 NEL
END
```

The PFSETP function must be called once for each panel which has been generated by **pangen04**. Do NOT call it more than once. Do NOT call it for dynamically loaded panels. The arguments are pointers to the data area, the xa area, the fcb table, and the pcb.

```
      INTERFACE TO SUBROUTINE PFSETP [C]
     +    (DLOC,ELOC,FLOC,GLOC)
      INTEGER*4 DLOC,ELOC,FLOC,GLOC
      END
```

The PAMSGE subroutine displays a null-terminated string on the defined message line

```
INTERFACE TO SUBROUTINE PAMSGE [C] (PLOC)
INTEGER*4 PLOC
END
```

The PAANSW function reads an answer into the tail of the previous message string, using the passed entry attributes.

```
INTERFACE TO INTEGER*2 FUNCTION PAANSW [C] (PLOC,ATT)
INTEGER*4 PLOC
INTEGER*2 ATT
END
```

The PAFUNC subroutine sends the sequence for the requested screen function:

1. Initialisation
2. Termination
3. Clear Screen and Home
4. Erase to end of line
5. Backspace cursor
6. Bell

7-10. Spare

```
INTERFACE TO SUBROUTINE PAFUNC [C] (CFUNC)
INTEGER*2 CFUNC
END
```

The **PACINP** function returns the value of a single keyboard character

```
INTERFACE TO INTEGER*2 FUNCTION PACINP [C]
END
```

The **PAROUT** subroutine writes a single character to the screen, with no highlighting. Use **PAROUT**(ICHAR(C)).

```
INTERFACE TO SUBROUTINE PAROUT [C] (CCH)
INTEGER*2 CCH
END
```

The **PAEOUT** subroutine writes a single escape sequence character to the screen, assumed not to move the cursor. Use **PAEOUT**(ICHAR(C)).

```
INTERFACE TO SUBROUTINE PAEOUT [C] (CCH)
INTEGER*2 CCH
END
```

The **PACSTA** function returns non-zero if a character is ready at the keyboard. Note that in some PANEL environments, this function always returns true.

```
INTERFACE TO INTEGER*2 FUNCTION PACSTA [C]
END
```

The **PACOUT** subroutine writes a single character to the screen, using any highlighting currently in effect. Use **PACOUT**(ICHAR(C)).

```
INTERFACE TO SUBROUTINE PACOUT [C] (CCH)
INTEGER*2 CCH
END
```

The **PAGETF** subroutine gets the codes corresponding to a single key depression from the keyboard, and returns either a function number or a data character. We suggest saving the result in an INTEGER*2 value equivalenced to a two-character array. The first element of the array is the value of the keyfunction or if that is zero then the second element is the character.

```
INTERFACE TO INTEGER*2 FUNCTION PAGETF [C]
END
```

The **PACURS** subroutine sets the cursor position for input, based on a top-left position of (0,0).

```
INTERFACE TO SUBROUTINE PACURS [C] (CX,CY)
INTEGER*2 CX,CY
END
```

The **PASBAD** subroutine sets the cursor position for output, based on a top-left position of (0,0).

```
INTERFACE TO SUBROUTINE PASBAD [C] (CX,CY)
INTEGER*2 CX,CY
END
```

The **PAHION** subroutine sends the on sequence for the current highlight type, using the highlight modifier if specified.

```
INTERFACE TO SUBROUTINE PAHION [C] (CMODI)
INTEGER*2 CMODI
END
```

The **PAHIOF** subroutine sends the off sequence for the current highlight type.

```
INTERFACE TO SUBROUTINE PAHIOF [C]
END
```

The **PAHTYP** function sets the highlight type and returns its corresponding method (which may be augmented by 128).

```
INTERFACE TO INTEGER*2 FUNCTION PAHTYP [C] (TYPE)
INTEGER*2 TYPE
END
```

The **PAPAGE** subroutine sets the active display page and the working page number for IBM PC Colour/graphics adapter.

```
INTERFACE TO SUBROUTINE PAPAGE [C] (ACT,WKG)
INTEGER*2 ACT,WKG
END
```

The **PACHCL** subroutine changes the background colour for subsequent clear screen calls and the border colour, both for the current working page of the IBM PC Colour/graphics adapter.

```
INTERFACE TO SUBROUTINE PACHCL [C] (CLS,BRD)
INTEGER*2 CLS,BRD
END
```

The **PASOUT** subroutine displays a character string using the **PAROUT** subroutine. The string must be null-terminated

```
INTERFACE TO SUBROUTINE PASOUT [C] (PLOC)
INTEGER*4 PLOC
END
```

The **PALIFT** function lifts and saves a screen block into a provided buffer.

```
INTERFACE TO SUBROUTINE PALIFT [C] (X,Y,L,H,PLOC)
INTEGER*2 X,Y,L,H
INTEGER*4 PLOC
END
```

The **PARDSP** function redisplays a screen block from a provided buffer.

```
INTERFACE TO SUBROUTINE PARDSP [C] (X,Y,L,H,PLOC)
INTEGER*2 X,Y,L,H
INTEGER*4 PLOC
END
```

The **PANELF** function performs screen operations using a field control block (normally defined as a character array).

```
INTERFACE TO INTEGER*2 FUNCTION PANELF [C] (FUNC,FCBLOC)
INTEGER*2 FUNC
INTEGER*4 FCBLOC
END
```

The **PAMENU** function returns a menu choice using a multiline field as a menu.

```
INTERFACE TO INTEGER*2 FUNCTION PAMENU [C] (FCBLOC,MHT,MHM,STL)
INTEGER*4 FCBLOC
INTEGER*2 MHT, MHM, STL
END
```

The **PAUSEP** function activates a panel control block.

```
INTERFACE TO INTEGER*2 FUNCTION PAUSEP [C] (PCBLOC,STF)
INTEGER*4 PCBLOC
INTEGER*2 STF
END
```

The **PANELS** function performs a full-screen operation.

```
INTERFACE TO INTEGER*2 FUNCTION PANELS [C] (FUNC)
INTEGER*2 FUNC
END
```

The **PASCNL** function returns a pointer and so cannot be called directly from FORTRAN. We use **PFSCNL** instead.

```
INTERFACE TO SUBROUTINE PFSCNL [C] (LNO, PLOC)
INTEGER*2 LNO
INTEGER*4 PLOC
END
```

The **PALOAD** function loads a panel from disk, using the panel name preloaded into the passed panel control block.

```
INTERFACE TO INTEGER*2 FUNCTION PALOAD [C] (PCBLOC,FSW,DSW)
INTEGER*4 PCBLOC
INTEGER*2 FSW, DSW
END
```

The **PAFREE** function frees a dynamically loaded panel.

```
INTERFACE TO INTEGER*2 FUNCTION PAFREE [C] (PCBLOC)
INTEGER*4 PCBLOC
END
```

The **PAMLOP** function opens a multiple panel file and index using the passed null-terminated filenames.

```
INTERFACE TO INTEGER*2 FUNCTION PAMLOP [C] (MLOC,ILOC)
INTEGER*4 MLOC,ILOC
END
```

The **PALOOK** function searches for the panel name of the passed pcb in the index, and prepares to read it.

```
INTERFACE TO INTEGER*2 FUNCTION PALOOK [C] (PCBLOC)
INTEGER*4 PCBLOC
END
```

The **PAMLCL** function closes the multiple panel file.

```
INTERFACE TO INTEGER*2 FUNCTION PAMLCL [C]
END
```

The **PFGET** function is used to retrieve C variables for use in your FORTRAN program. Variables are referenced by their code number in the list below, and the use of a *LOC* pointer allows a variable number of bytes to be returned. **PFGET** gets variable number n into the passed location. The pointer *PLOC* must point to a big enough area for the data type concerned. The table shows the code, the variable name, and the number of bytes transferred.

```
1       PAFLX           2
2       PAFLXS          2
3       PASET           256
4       PANFL           2
5       PAMULT          2
6       PAPSLO          2
7       PAPSHI          2
8       PAOVTH          2
9       PAMLN           2
10      PAMEN           2
11      PAMHD           max 80
12      PAMFCB          36
13
14      PARDHI          2
15      PAECB           48
16      PAOLDH          2
17      PASSEC          2
18      PASCRO          2
19      PABRD1          9
20      PABRD2          9
21      PAKTS           2
22      PVWSTR          127
23      PAXTRA          17
24      PAVAL           80
25      PAVLN           20
26      PVINCH          1
27      PVDINCH         1
28      PVNINCH         1
29      PVKARG          2
30      PVLARG          2
31      PAFSVN          2
32      PAXASN          2
```

```
INTERFACE TO SUBROUTINE PFGET [C] (VARNO, PLOC)
INTEGER*2 VARNO
INTEGER*4 PLOC
END
```

The **PFPUT** function is used to set a C variable using a value in your FORTRAN program. It puts a value into variable number n from the passed location. The pointer *PLOC* must point to a big enough area for the data type concerned. The variable numbers are the same as for **PFGET**.

```
INTERFACE TO SUBROUTINE PFPUT [C] (VARNO, PLOC)
INTEGER*2 VARNO
INTEGER*4 PLOC
END
```

The **PFGETI** function is similar to **PFGET**, but is only defined in the 'I' library for memory-mapped operation on an IBM Personal Computer. It retrieves variables which are only defined in that library. The code numbers used to identify each variable are shown below. **PFGETI** gets variable number n into the passed location. The pointer *PLOC* must point to a big enough area for the data type concerned. The table shows the code number, the variable name, and the number of bytes.

1	PAITRM	11
2	PAIMAT	16
3	PAICAT	16
4	PAIATT	16
5	PAIBOR	1
6	PAIEND	1
7	PAIBEL	2
8	PAICFM	4
9	PAIDL2	2
10	PAIPFX	4
11	PAIIGP	4
12	PAIIGN	32
13	PAIKEY	65

```
INTERFACE TO SUBROUTINE PFGETI [C] (VARNO, PLOC)
INTEGER*2 VARNO
INTEGER*4 PLOC
END
```

The **PFPUT** function is used to set an 'I' library C variable using a value in your FORTRAN program. It puts a value into variable number n from the passed location. The pointer *PLOC* must point to a big enough area for the data type concerned. The variable numbers are the same as for **PFGETI**.

```
INTERFACE TO SUBROUTINE PFPUTI [C] (VARNO, PLOC)
INTEGER*2 VARNO
INTEGER*4 PLOC
END
```

The **PFGETD** function gets the entire data area for a field into the passed location. It is used when code has not been generated for the panel (ie the panel has been dynamically loaded) or when there is a specific reason to refer to the field by number or position rather than by name. The field is referenced by its fiel control block passed to this function. The size of the data returned is calculated from the data are width and height in the passed fcb. The pointer *PLOC* must point to a big enough area.

```
INTERFACE TO SUBROUTINE PFGETD [C] (FCBLOC, PLOC)
INTEGER*4 FCBLOC
INTEGER*4 PLOC
END
```

The **PFPUTD** function puts data into the data area for a field from the passed location. It does the opposite of **PFGETD** described above.

```
INTERFACE TO SUBROUTINE PFPUTD [C] (FCBLOC, PLOC)
INTEGER*4 FCBLOC
INTEGER*4 PLOC
END
```

The **PFGETF** function gets the entire field control block for field number n into the passed location. The function will transfer 36 bytes for a full fcb and 24 bytes for an abbreviated fcb. The pointer PLOC had better point to a big enough area. If the field number is zero the current field **PACRF** is retrieved, which will only be of significance during validation operations.

```
INTERFACE TO SUBROUTINE PFGETF [C] (FNO, FCBLOC)
INTEGER*2 FNO
INTEGER*4 FCBLOC
END
```

The **PFPUTF** function puts data into the fcb area for field number n from the passed location. As with **PFGETF**, the amount of data transferred will depend on whether the fcb to be moved has the 'abbreviated' attribute: however the function will not allow you to transfer a full fcb into an area previously containing an abbreviated fcb. Instead the abbreviated indicator will be set and only 24 bytes will be transferred. If the field number is zero the current field fcb **PACRF** is filled.

```
INTERFACE TO SUBROUTINE PFPUTF [C] (FNO, FCBLOC)
INTEGER*2 FNO
INTEGER*4 FCBLOC
END
```

The **PFXSCN** function gets an extended attribute string or substring for a field. The first argument is a pointer to the fcb. The second is the extended attribute number (0..15). The third is either -1 to return a complete extended attribute string or a substring number (0..31) to return a substring. The data is returned to the result area specified by the fourth argument, and a maximum number of characters specified by the fifth argument

is transferred. If nothing is transferred, the function returns -1. If a scan for substrings is done, it returns the total number found. Otherwise, it returns zero. Note that no corresponding function is provided to put data into an extended attribute string.

```
      INTERFACE TO INTEGER*2 FUNCTION PFXSCN [C]
     +  (FCBLOC,XANO,SSNO,RESLOC,MAXN)
      INTEGER*4 FCBLOC, RESLOC
      INTEGER*2 XANO, SSNO, MAXN
      END
```

The **PFPUTP** function specifies that a new function pointer is to be placed in the validation table. The first argument is the type of validation function (*CV*=0, *KV*=1, *LV*=2, *FV*=3, *XV*=4). The second is the entry number in the table, and the third is a function pointer. The validation function may either be a C function, or a FORTRAN (or Pascal) function with a C attribute. The returned value is the previous contents of the pointer table entry, which can be saved and subsequently restored with a further call to **PFPUTP**.

```
INTERFACE TO INTEGER*4 FUNCTION PFPUTP [C] (TYPE,ENO,FLOC)
INTEGER*2 TYPE, ENO
INTEGER*4 FLOC
END
```

The **PABM** subroutine is a quick-and-dirty block mover. It is passed pointers to source and destination areas, and a number of bytes to move. No checking is done.

```
INTERFACE TO SUBROUTINE PABM [C] (SRCLOC,DESTLOC,BYTES)
INTEGER*4 SRCLOC,DESTLOC
INTEGER*2 BYTES
END
```

The **PABF** subroutine is a quick-and-dirty block filler. It fills a block of data of the specified length with the character specified. The fill character should be passed via *ICHAR*.

```
INTERFACE TO SUBROUTINE PABF [C] (DESTLOC,BYTES,FILL)
INTEGER*4 DESTLOC
INTEGER*2 BYTES
INTEGER*2 FILL
END
```

The **strlen** function is in the *CEXEC* library. We use it in the numeric validation function, so define it here.

```
INTERFACE TO INTEGER*2 FUNCTION STRLEN [C] (PLOC)
INTEGER*4 PLOC
END
```

Generating FORTRAN Code

The Code Generator **pangen04** generates two separate files. *pname.INC* is designed to be included in your program to specify the common blocks and data fields. *pname.FOR* is a BLOCK DATA subprogram which is compiled and linked with your application. The *.FOR* module no longer includes the *.INC* file.

In the *.INC* file, only those fields for which a name has been specified by the **panel** editor have equivalence statements generated. This helps to ensure that the compiler does not run out of equivalence table space.

The code generator now has three command line switches as follows:

-F Generates 'full' field control blocks. The default is 'abbreviated'.

-V Turns off verbose mode. **PANEL.VDU** is not required.

-Nx Changes default field names starting with 'F' to 'x'.

Linking PANEL programs

Your linkage command file should name as input files your application program modules and any generated (and compiled) panel descriptions. The library specification should be in the following sequence:

PANEL04I (if you are using the memory-mapped option)
PANEL04 The main panel library
CEXEC From the MS-FORTRAN distribution disk

The remaining FORTRAN libraries will normally be searched automatically. As an example, **test04** is linked as follows:

```
PANGEN04 -FV DEMO
LINK TEST04+DEMO,,,PANEL04I+PANEL04+CEXEC
```

Important Note. The generated code for a panel consists only of a BLOCK DATA subprogram for which the common blocks are defined in the header file. This means that you will normally get no linkage error if you accidentally omit the corresponding object file from your link list. This behaviour differs from the other PANEL language interfaces. Running the program with empty common blocks where there should be field control blocks and data areas can be quite an interesting experience. This problem does not affect programs where all the panels are dynamically loaded.

Modifying the Supplied Validation Functions

If you have the Microsoft C Compiler, version 3, you can recompile any of the C functions as necessary. This discussion assumes that you do not have the C compiler.

The validation functions are controlled by the Validation Pointer Table in module **PNLFVPTC.C**. The table can (and does) contain pointers to both C functions and FORTRAN functions. Many of the functions are not complex, and could be replaced quite easily with modified FORTRAN functions if required. Any FORTRAN function should be specified with the '[C]' attribute.

Any new or replacement validation functions can be added to your list of files to be linked and will replace the function in the library with the same name. If you have a library manager you can replace the function in the library.

If you add new validation functions you will either have to recompile the **PNLFVPTC.C** function having replaced one of the dummy **PARET0** entries in the table, or use the **PFPUTP** function to specify the name of your new function. If you run out of dummy entries you will have to recompile the module to extend the tables.

The **PFPUTP** function can also be used to dynamically replace a validation function for certain stages of your program. When called with a new function pointer it returns the previous contents

of the table entry, which can be saved and restored to revert to normal operation.

A FORTRAN validation function can access its arguments using the same variables as described earlier in this supplement, but will need to use the **PFGET** function to retrieve them. See the provided **PNLNUMVF** and **PNLXV02F** modules for examples.

Low-Level Routines

The Low-Level Routines are provided in C source form. Most of the modifications required to these modules concern the *PAI..* variables for the memory-mapped option, and these can be retrieved and modified using **PFGETI** and **PFPUTI**. The low-level routines also include the **PATB** function included in the old Panel FORTRAN interface. This is called as **PATB**(TARGET,BITNO) and returns a LOGICAL*2 .TRUE. result if the bit is set. The target can be up to 32 bytes and the bit number up to 256. Bits are numbered from 1 starting at the left of the target. This function is written in assembler and should not be declared with the [c] attribute.

Section IV
PANEL for MS-BASIC

Using PANEL with MS-BASIC

Introduction

This section describes the use of PANEL with Microsoft or IBM Basic under MS-DOS or PC-DOS.

Screen layouts for use with Basic are designed using the **panel** screen editor in the same way as for any other programming language. The only difference for Basic is that there is a field size limit of 255 characters for the Basic Interpreter. Because the DEMO panel has fields larger than this, we supply a DEMOB panel with smaller fields for use with Basic. If your application also needs larger fields, for example when designing a 'help' screen, you will need to divide the area into multiple fields in order to remain within this limit. When using extended attributes with the interpreter, each string must also be limited to 255 characters. The maximum number of extended attribute substrings is 32.

The Basic language interface does use any field names you may have assigned using **panel**, except as optional comments for the generated code. From Basic, it is necessary to refer to fields by number if they must be processed individually.

Before you can make any use of a screen layout from Basic, it is necessary to run the **pangenb** generator. This program can produce either a portion of a Basic program, containing string initialisation and other statements, or alternatively a data file containing details of the screen layout in a form which is easily read by Basic. Further details of **pangenb** are given below.

A Basic program communicates with PANEL by making calls to a module which is loaded below it in the DOS user RAM area. This module is loaded by running the program **pnlbas**. **pnlbas** can either load your own program for you, or run the Basic Interpreter; by default it simply loads another copy of COMMAND.COM to accept any DOS command you enter. When the loaded program terminates, **pnlbas** is also removed to free your RAM area. The process is described in more detail below. The PANEL Basic language interface requires DOS 2.0 or above to operate.

pnlbas is required to run your application programs written either with the Basic Interpreter or the Basic Compiler, but is not used by

any of the supplied PANEL system programs such as the **panel** screen editor and **pangenb**.

Running PANGENB

The command-line syntax for **pangenb** is as follows:

PANGENB [-options] panelname

The panel name should be entered without the 'PNL' file type, and any file type entered will be ignored. By default, the program requires the PANEL.VDU file to be available, and displays a screen showing the progress of the generation. The -v option suppresses this 'verbose' mode and **pangenb** then operates without PANEL.VDU. The PANGENB.MSG file is required.

Option letters may be entered separately, each preceded by a dash ('-'), or the letters may be entered together following a single dash. Options may be entered in upper or lower case. In 'verbose' mode, if no panel name is entered the program will prompt you for it. Do not then re-enter any option letters you have already included on the command line: all letters act as 'toggles' and an even number of occurrences therefore has no effect. The effect of each option is described with the details of the generated output below.

If you habitually use **pangenb** with certain options, they may be predefined by entering the letters on line 9 of the PANGENB.MSG file, preceded by a space and a hyphen. Further option toggles entered on the command line or at the program prompt may be also be used. Only the letters on line 10 of the message file are valid.

The default output from **pangenb** is a data file with the same name as the panel name and a file type of 'BPN'. This file is in a format which can be read by Basic. The use of the '-b' option causes the program to generate a line-numbered Basic subroutine which can be edited into your program with a text editor or with the Basic 'merge' command. Line numbers start at 10000 unless a numeric value immediately follows the -b with no intervening space. This value then overrides the starting line number. The file type of the output file is 'BAS' when the -b option is used. WARNING: an existing file with the same name as the output file will be overwritten by **pangenb**.

Using PANEL with MS-BASIC

Using the PANEL Basic Language Interface

The first step is to run the **pnlbas** program which includes the code to perform all PANEL functions. The standard version of **pnlbas** requires PANEL.VDU to be available. If this file is not in the current DOS 2 directory, you may set up a 'PNL=' environment specifier to enable **pnlbas** to find the file. If PANEL.VDU is not available, **pnlbas** terminates immediately with an error message. The option of using the 'TERM=' environment specifier is not available for **pnlbas**.

A special version of **pnlbas**, named **pnlbasi**, is provided with versions of the PANEL Basic language interface for IBM PC compatible systems. When **pnlbasi** is used, the output from PANEL library functions is written direct to the screen buffer, resulting in substantially faster operation. **pnlbasi** can be configured using the environment specifiers described earlier in this supplement for IBM PC operation. Only use **pnlbasi** on a system which is 100% IBM PC compatible. If you are able to use **pnlbasi**, it can be used exactly as described for **pnlbas** in this document.

pnlbas will display the segment address at which it has loaded together with the offset of the first subroutine entry point, from which the remainder can be calculated. These values will be required in your Basic program to gain access to the PANEL facilities. The program will also write out a disk file named PSEG containing the values. If you read this file in your Basic program as shown in the examples, your application will automatically find the PANEL segment. The PSEG segment value can change as a result of operating system configuration changes, and the POFF offset may change in later releases of the PANEL Basic language interface.

The file named PSEG is always written in the current directory at the time **pnlbas** is run. If you subsequently change directory paths before running your Basic program it will be necessary to use a pathname when reading the PSEG file in your Basic program.

pnlbas operates in one of two ways depending on whether you enter a command line argument or not:

If the **pnlbas** command is entered on its own, **pnlbas** attempts to load a copy of COMMAND.COM to process additional commands. The copy of COMMAND.COM is found by reference to the COMSPEC environment variable, which is set up automatically by DOS at boot time to point to the directory containing the program. You can modify the COMSPEC parameter by means of the SHELL statement in the CONFIG.SYS file. In the absence of a COMSPEC environment specifier **pnlbas** looks for a file named COMMAND.COM in the current directory.

If the **pnlbas** command is followed by arguments, the first is taken as a program name to be loaded by **pnlbas**. The named program is found by reference to the PATH environment specifier, if any. If no PATH is found, the current directory is used. **pnlbas** searches first for an EXE file and then a COM file in each directory named. Any further arguments are passed to the loaded program.

f the program cannot be found or if there is insufficient memory to load it, **pnlbas** reports the problem and terminates with a return code of 1. When the loaded program terminates, **pnlbas** also terminates and frees all the RAM used. If the loaded program is COMMAND.COM, you can terminate it by typing EXIT at the command prompt. You can always type EXIT if in doubt as to whether **pnlbas** is still resident, as the 'bottom-level' copy of COMMAND.COM will ignore this command.

The program name entered on the **pnlbas** command line will often be that of the Basic Interpreter (eg BASICA, MSBASIC, GWBASIC or BASIC86). Alternatively you may enter the name of your own Compiled Basic program. WARNING: loading a program which terminates and stays resident is not recommended. Examples of entries are given below:

PNLBAS	- loads COMMAND.COM to process further commands.
PNLBAS MYCOMP	- loads and runs a compiled Basic program called MYCOMP.
PNLBAS BASICA	- loads IBM Basic Interpreter.

PNLBAS BASICA MYPROG - loads and runs an interpreted Basic program called MYPROG.

Note that the present version of **pnlbas** is not re-entrant, and therefore cannot be called from multiple concurrent tasks. Also, if you patch or poke the **pnlbas** segment the effects of the change will remain until **pnlbas** is removed from memory, not just until your program finishes. After **pnlbas** has been installed in memory, the PANEL system may be called from your Basic program as described below. Either interpreted Basic or compiled Basic may be used, with the minor code changes described below.

PANEL Generated Code

The **pangenb** program with the '-b' option generates a complete Basic subroutine which consists of the following elements:

1. REM statements documenting the date and time of generation.
2. Five dimension statements for the string arrays described below. When using the Interpreter, if you need to call the full generated subroutine more than once, you should call the line number following these dimension statements on the second and subsequent occasions.

   When the generated code is to be compiled, it is necessary for at least the data area DIM statements to be moved out of the subroutine and placed at the start of your main program, so that the data fields can be referenced in the program without the compiler reporting a dimension error. This move can be made with your text editor, or by a suitable sequence of Interpreter source file commands.
3. An assignment statement to set the message line for the panel to the correct value. The integer variable name is formed from the letters 'ML' followed by the panel name.
4. Two assignment statements to set the counts of 'input' and 'output' fields to the correct values. The integer variable names are formed from the letters 'IC' and 'OC' followed by the panel name.
5. A set of assignment statements to initialise string variables

for all of the input data areas for the panel. By default, these are defined as string expressions so that a subsequent call to the generated subroutine will reset them to their initial values. If you specify a '-i' option for **pangenb** the strings will be defined as string constants, so that data entered on the screen actually modifies the program text. This option is provided only to save dynamic string space in special cases. Above a specific field size, strings are always generated line-by-line as a string expression.

6. A set of assignment statements to initialise string variables for all of the 'no-input' fields. The option is the opposite way round to the input fields: '-o' as a **pangenb** option is used to force the use of string expressions, and the default is to generate string constants except for large multi-line fields.

7. Two sets of assignment statements to initialise string constants for the control information for each field. As for the data areas, input and 'no-input' fields are placed in separate arrays. The control information consists of eleven semi-colon separated numeric elements which initialise the field control block for each field. Using the nomenclature of the PANEL reference manual (pages 79 to 81) the elements are SX, SY, SL, SH, SF, SA, SM, HIGHTYPE, HIGHMODI, CHARVALN, and LINEVALN. Other fields are not used by the Basic Language interface. If the SA field has the 'extended attribute' bit set, then two additional numeric elements appear in the string. The first is a bit field with one bit set for each of the sixteen possible extended attributes present for the field. The second is the index in XAx$ of the first extended attribute string for the field.

8. Assignment statements to specify the extended attribute strings for fields in the panel. Each string is generated with a terminator character of 'Z', which will automatically be replaced by a null so that the strings can be more easily processed by the 'C' language routines in the interface.

9. DEF SEG and CALL statements with the appropriate parameters for the **PAUSEP** function described below. Having assigned all the string and integer fields the program has to pass their addresses to the PANEL run-time module so that subsequent function calls can use the defined panel. The

call is followed by a RETURN statement so that the application program can GOSUB to the generated code.

If the '-c' option is used in the **pangenb** command line, the generated code has field names added as comments on each of the control string initialisation statements, together with comments identifying the generated numeric assignment statements.

Double quote characters in data areas are changed to single quotes by **pangenb** and a warning message is displayed. A warning is also displayed if a field data area is more than 255 characters long. The warning may be ignored if the panel details are only to be compiled, but after this warning DO NOT attempt to use the output of the generator with the Basic Interpreter.

Generated Data Files

Without the '-b' option the **pangenb** program generates all of the information described in the preceding section but in the form of data values in a file. The file contains the ML, IC, and OC values, plus a count of the number of extended attribute stings, followed by five arrays of strings. It is necessary to use the numeric values to determine how many strings to read. Examples showing how to read the data file will be found in the supplied source files.

Subroutine Library Calls

The Basic Language subroutine library calls are designed to correspond as closely as possible to those for the other languages described in the PANEL reference manual. There are of course some omissions, and the principal difference is that the low-level functions are all accessed through a single call point (**PANLOW**) instead of having a separate entry for each function.

For the Interpreter, the segment and offset values for each call can be obtained by reading the PSEG file which contains two hexadecimal values. The first is the value to be used for the DEF SEG statement, and the second is the value of the offset for **PSETUP**. The other offsets can be obtained by successively adding three to this value. Examples of this process can be found in the sample source files.

When using the compiler, the segment and offset values PSEG and POFF have to be passed to **PSETUP** so that the other calls can be addressed correctly. The linkage between your program and **pnlbas** is handled by a small object module named PNLBC.OBJ, which must be linked with your program.

The syntax and function of each call is described in the following sections, which should be read in conjunction with the PANEL reference manual.

CALL PSETUP(B%) / CALL PSETUP(PSEG,POFF)

This call must be the first **pnlbas** call made in any program. The first form is for the Interpreter, and B% MUST have a value of 1. The second form is for use with the Compiler, and allows the link module PNLBC to access the **pnlbas** module.

If **PSETUP** is not called with the correct value(s) before any other ANEL subroutine call, unpredictable results will occur.

CALL PANLOW(FUNC%,A1%,A2%,RET%)

The **PANLOW** call is used for the 'low-level' functions described in the PANEL manual. The call is coded for each function as described below:

Function	F%	A1%	A2%	RET%
PAFUNC	(1-10)	0	0	-
PACINP	11	0	0	(input char)
PAROUT	12	(char)	0	-
PACSTA	13	0	0	(status)
PACOUT	14	(char)	0	-
PAGETF	15	0	0	(char/func)
PACURS	16	X	Y	-
PASBAD	17	X	Y	-
PAHION	18	(method)	0	-
PAHIOF	19	0	0	-
PAHTYP	20	(type)	0	(method)
PAPAGE	21	(act)	(disp)	-
PACHCL	22	(cls)	(bord)	-

Character values in the above list must be passed as the numeric ASCII equivalent.

CALL PAUSEP(MLx%,ICx%,OCx%,IDx$(1),ODx$(1),IBx$(1), OBx$(1),XAx$(1))

This is the main call required to set up a panel for use by the calls listed below. The arguments will normally be as shown above, with the 'x' representing the panel name, and the call will normally be generated by **pangenb**, so that you do not have to remember all of the parameters. When the 'dynamic load' option is used and the panel details are read from a 'BPN' data file, the variable names used in the **PAUSEP** call will be those into which the data has been read by your program.

The argument definitions are as follows:

MLx%	Message line for use by PAMSGE and PAANSW
ICx%	Count of input fields
OCx%	Count of output fields
IDx$	First element of input field data array
ODx$	First element of 'no-input' field data array
IBx$	First element of input field control array
OBx$	First element of 'no-input' field control array
XAx$	First element of extended attribute array

No error code is returned by this function; however the RET% parameter for **PANELF** and **PANELS** described below will have a value over 250 if inconsistencies are detected in the data, such as the wrong number of control elements, or a data area size not matching the size calculated from the control block.

Note that if a screen panel layout has either no input fields or no 'no-input' fields, the appropriate count value should be zero but dummy string variables must be inserted in the CALL statement.

The function of this call is to pass addresses to **pnlbas** for use with subsequent calls. This technique is designed to simplify the other calls and reduce program size. WARNING: The Basic Interpreter can move arrays about when new variables are defined, and so it is

vital to ensure that **PAUSEP** is called again if there is any risk of this having happened. The usual symptom of this problem is that **PANELF** returns an error code over 250, or **PANELS** fails to display or read the expected number of fields. In the generated code, the **PAUSEP** statement is followed by a RETURN, so it is easy to GOSUB to it.

CALL PANELF(FUNC%,FLDN%,RET%)

This call is used to request all of the PANEL functions for an individual field. The FUNC% is a numeric argument which corresponds to the command number shown under '**PANELF**' in the panel reference manual. Because input and 'no-input' fields are held in separate arrays, each set is numbered from 1; however the **PANELF** call treats the FLDN% argument as if the 'no-input' fields were numbered up from the input fields. If there are seven input fields, 'no-input' field number 4 is referred to as field 11 when calling **PANELF**. This numbering method is less confusing than it might appear because it is very seldom that one wishes to process individual 'no-input' fields.

During data entry the standard validation features described under 'attributes' and 'extended attributes' in the PANEL reference manual are available, with the exception of the 'NF' extended attribute. There is no facility in the PANEL Basic language interface to add to the supplied validations; however their operation may be modified in certain ways by using the service function **PASERV** described below.

CALL PANELS(FUNC%,RET%)

This function performs full screen operations, and again uses the function numbers described under '**PANELS**' in the PANEL reference manual. Values for FUNC% above 100 operate on either the input or the 'no-input' fields: those in the range 1-40 perform the corresponding field function on EVERY field.

CALL PAMSGE(MSG$)

The **PAMSGE** call displays the passed message on the defined message line for the current panel. If you plan to use the **PAANSW**

function it will normally be necessary to extend the message string either with blanks or a default answer before displaying it. The message end point is not available to the calling program.

If the first character in the passed string is a colon, the remainder of the string is used as the 'message heading' as described in the PANEL manual. To clear the message heading it is necessary to call the **PAMSGE** function with a string containing just a colon.

If the passed message string is empty, message line is cleared. The special message string "0" causes the previous message to be repeated. Examples of the use of the message and answer functions appear in the supplied source files.

CALL PAANSW(ANS$,SAT%,RET%)

This routine gets an answer to the previously displayed message. The ANS$ string should contain the default reply, which has been displayed at the end of the message; the length of the ANS$ string defines the length of the reply. The answer attribute field contain the attributes to be used for the answer entry, which are defined as the sum of the relevant values in the following list:

Upper Case	&H0080
Echo	&H0040
CR Reqd	&H0020
Wipe	&H0010
RJ	&H0008
Num	&H0004
$	&H0002
USR	&H0001

Details of the processing of the attributes will be found in the PANEL reference manual. The RET% value returns the PANELF return code for the read of the answer, and can be used to detect function key usage during answer entry.

CALL PASCNL(LINE%,SCRNL$)

The **PASCNL** function operates exactly as described in the PANEL reference manual. It returns to SCRNL$ a the line of the current

screen display which is defined by the LINE% argument. If the line number is negative or greater than the maximum row number, a line of hyphens is returned. If the size of SCRNL$ at the time of the call is less than the screen width then the returned line will be truncated; if larger, then each line is filled on the right with blanks.

CALL PASERV(FN%,ARG1%,ARG2%,ARG3$)

The **PASERV** function performs miscellaneous service functions described below. It must always be called with all four arguments even if not all are required for a particular function. The function to be performed is selected by means of the FN% parameter. You should probably not use these functions until you have acquired some familiarity with the PANEL Basic interface. They are designed to reduce the need to PEEK and POKE into **pnlbas**. The values of the FN% parameter from 1 to 9 select the following actions:

1 - The passed value in ARG1% is placed into the **PARHDI** variable. This value should be a highlight type number which will be used to highlight each field read as the cursor is moved to it.

2 - The passed value in ARG1% is placed into the **PAMLN** variable. This resets the message line. The message line is also set by each call to **PAUSEP**.

3 - The passed value in ARG1% is placed into the **PAHP150** variable. The value defaults to zero and should be set to 1 to enable touch screen support for the Hewlett Packard HP150. During a screen read, the cursor will move to any entry field which is touched. If a value greater than one is placed in this variable, a field with or height of one will be increased to the given value in the touch field definition, so that it is easier to select the field.

4 - The offset address in PSEG of **PAECB** is returned in ARG1% and the offset address in PSEG of **PAVDU** is returned in ARG2%. These values may be used by advanced users to access the PANEL environment control block and VDU area. The **PAVDU** area is not used after initialisation in the memory-mapped version **pnlbasi**.

5 - The current contents of the **PAXTRA** string are returned to the ARG3$ string if ARG1% is zero, and replaced by the ARG3$ string if ARG1% is non-zero. **PAXTRA** may contain up to eight pairs of characters in the format 'aA...' for uppercase conversion of alphabets containing other than the letters A..Z.

6 - The current contents of the **PAVAL** string are returned to the ARG3$ string if ARG1% is zero, and replaced by the ARG3$ string if ARG1% is non-zero. **PAVAL** may contain up to 63 characters which are used to validate input for character validation routine number 1.

7 - The current contents of the **PAVLN** string are returned to the ARG3$ string if ARG1% is zero, and replaced by the ARG3$ string if ARG1% is non-zero. **PAVLN** may contain up to 12 characters which are used to validate numeric input.

8 - The current contents of the **PAHPBUF** string are returned to the ARG3$ string if ARG1% is zero, and replaced by the ARG3$ string if ARG1% is non-zero. **PAHPBUF** may contain up to 40 escape sequence characters which are used to establish touch areas for each field. The string must contain five conversion specifiers in 'C' language printf format. Full details appear in the source files distributed with PANEL 'C' language interfaces.

9 - Not currently implemented.

10 - The current contents of the **PAFLX** variable are returned to the ARG2% variable if ARG1% is zero and replaced by the ARG2% variable if ARG1% is non-zero. **PAFLX** can be set to the field number at which full-screen read is to start and the value returned after a read is the saved value of **PAFLX** at the time the read was finished. **PAFLX** itself is zeroised after every **PANELS** call.

11 - The passed value in ARG1% is placed in the **PAPSLO** variable and the passed value in ARG2% is placed in **PAPSHI**. These two variables define the range of field 'level' numbers processed by the **PANELS** routine.

12 - The passed value in ARG1% is placed in the **PAOVTH** variable. Fields with 'level' numbers higher than this value

will be processed in 'overlay' mode, that is the previous screen contents will be saved before display. A maximum of 16 save areas are available for such transient fields, and the saved details must eventually be freed either by the 'flose' or 'funsh' function of **PANELF** or **PANELS**.

13 - The passed value is ARG1% is used as the argument of a call to **PASCSE**. This function ('screen save enable') must be called before any call is made for overlay field or pop-up menu processing. After making this call you should ensure that the screen is cleared by using function number 3 of **PAFUNC**.

14 - The passed value in ARG1% is taken as a field number for which the **PAMENU** function is to be called. The passed value in ARG2% is the highlight type to be used for the moving menu bar. The result of the menu call is returned in ARG2%.

WARNING: Your program is responsible for resetting any values which you change; remember that the area being altered may remain resident after your program terminates.

Program Size

The current release of the **pnlbas** program occupies approximately 48kb. This memory requirement is 'outside' the Basic segments and therefore does not restrict program size in any way provided that the total RAM available is sufficient. Note that the PNLBAS.EXE file contains initialisation code which is overlaid by the loaded program.

Example Files

The sample files provided are as follows:

DEMOB.PNL	Demonstration panel file for use with Basic
DEMOB.BAS	Program file generated by **pangenb** from DEMOB.PNL
DEMOB.BPN	Data file generated by **pangenb** from DEMOB.PNL
TESTBI.BAS	Displays and reads DEMOB panel (Int)
TESTBC.BAS	Displays and reads DEMOB panel (Comp)

Using PANEL with MS-BASIC

PANTSTBI.BAS Displays and reads any panel dynamically (Int)
PANTSTBC.BAS Displays and reads any panel dynamically (Comp)

To use the files, follow the steps below:

TESTBI

PNLBAS

Load the Basic Interpreter (BASICA or MSBASIC)

LOAD "TESTBI"
MERGE "DEMOB"
RUN

TESTBC

Load the Basic Interpreter (BASICA or MSBASIC)

LOAD "TESTBC"
MERGE "DEMOB"
Move lines 10030 to 10070 as indicated in TESTBC.BAS

SAVE "TEST",A
SYSTEM

BASCOM TEST;
LINK TEST+PNLBC;

PNLBAS TEST

PANTSTSBI

PNLBAS BASICA PANTSTBI (PC-DOS)

PNLBAS MSBASIC PANTSTBI (MS-DOS)

PANTESTBC

BASCOM PANTSTBC/E;
LINK PANTSTBC+PNLBC;

PNLBAS PANTSTBC

Section V
PANEL for Amiga

Using PANEL on the Amiga

Introduction

This section provides additional documentation for PANEL for the Commodore Amiga. PANEL is available in both native Amiga versions and also for cross-development under MS-DOS or PC-DOS. Unless otherwise noted, the description below applies to all versions. The PANEL interface number is 'C1' for the Lattice versions, 'C2' and 'C3' for the Manx versions (16 and 32 bit intergers respectively).

The versions of PANEL for the Amiga are designed to allow 'traditional' text-based data-entry applications to be written for the Amiga without having to get involved in the intricacies of the Intuition/AmigaDOS interface. Programs written with PANEL for other environments should be able to be ported to the Amiga simply by recompilation using the native or cross-compiler versions of C. Although applications can be developed easily and quickly using PANEL, they are able to use the advanced features of the Amiga and are fully integrated into the multitasking environment.

Screen layouts created using PANEL are generally portable between all supported environments. The only exception will be if you have included extended character set values in your screen designs, which may display differently when you move the application to the Amiga. Code generated by **pangenc** also needs to be regenerated for use on a machine with a different sequence of bytes in a word.

Function calls to the PANEL library are unlikely to require any change for the Amiga. Because the Amiga does not require the PANEL tailoring system, it will correspond most closely to the memory-mapped option in the IBM Personal Computer versions (6.01 and later) of PANEL.

Further details of portability considerations are given below.

Installation and Tailoring

IMPORTANT: PANEL programs, and applications created using unmodified PANEL library routines, look for certain parameter files on a device named *Panel:*. Since the native Amiga version of

PANEL uses this as a volume name, the programs will work without requiring any *assign* statement. If you use our programs, or those you have developed, on a disk with a different name, you must assign *Panel:* as a logical name to the appropriate physical volume. Alternatively, you may prefer to modify the low-level module PNL1Cx5C.C to use predefined parameters and avoid reading the parameter files.

The **tailor** and **keyboard** programs described in the PANEL reference manual are not used with the Amiga, and the PANEL.VDU file is also not required. Customising the PANEL screen and keyboard is done either by modifying the supplied low-level source module PNL1Cx5C.C or by using the configuration files *pnlcolors*, *paiatt*, or *pnlmode*. The *Panel:pnlcolors* file contains the Amiga colour register values (R,G,B) for up to sixteen registers, which will be loaded as each program is initialised. This file will normally contain eight colour register entries to correspond to the default screen depth of 3, and the file can then be set up interactively using the **pcmu** program.

The *Panel:paiatt* file contains up to sixteen pairs of numeric values giving the foreground and background colour register number to correspond to the sixteen PANEL highlight types.

The *Panel:pnlmode* file contains single-letter codes to select various screen modes.

All three files are sought in the device *Panel:*, which must either be a physical volume name, or must be set with an *assign* statement, preferably in your file *s:startup-sequence*. If you forget to make this assignment, the program may appear to hang after opening the screen. To escape from this situation, press 'Amiga-N' to redisplay the workbench screen and then 'cancel' the error requesters. On returning to the PANEL screen with 'Amiga-M' you will find that the program is working correctly, probably with unusual colours.

The keyboard assignments are predefined for the supplied PANEL utility programs (see the table below). In application programs you can modify the defaults by altering the *PAIKEY* table in the supplied source module PNL1Cx5C.C.

Using PANEL on the Amiga

Getting started: Native version

PANEL is delivered on two Amiga diskettes: the first contains the utility programs for set-up and program development and the second contains the PANEL library and the source and header files. Make a backup copy and a working copy of each diskette.

The working copy of the first disk can be used to set up colours, create and test screen layouts, and generate code as required which can be transferred to the diskette on which you compile programs. The header files and library file from the second disk will also be required for compiling and linking programs.

After setting the logical device *Panel:* as described above, you can start the **panel** program either from the CLI or using the icon. We recommend experimenting with **panel** and **pantest** using the DEMO panel as described in the main PANEL introduction manual. Your own screens can be designed with **panel** on the work diskette, or transferred from another system and modified as necessary. When you have used **pangenc** to generate the source code for your screen layout the resulting *.c* and *.h* files can be copied to your application work disks for compilation. You will need the header files from the PANEL distribution diskette for this step.

Most of the PANEL source files are validation source files which you will not need until you wish to implement your own validation routines. We recommend that you print and study carefully the low level module pnllcx5c.c, because this controls the interface between PANEL, AmigaDos, and Intuition. Use the sample source files for **pantestc**, **testc** and **pd** as models for your own programs.

Getting started: Cross version

The cross-compiler version of PANEL for Amiga is supplied in the same form as other PANEL MS-DOS C interfaces.

The **pcmu** program is supplied in executable form, and should be copied to the Amiga to allow the setting up of colours. The supplied *pnlcolors* configuration file should also be transferred to the Amiga and modified as required using **pcmu**.

Note that the supplied PANGENC.MSG file is pre-set for use when the byte-sequence of the target machine is different from that of the host. It does not require the '-s' switch because this is predefined on line 9. Specifying '-s' on the command line will toggle the switch off.

The Amiga Implementation

PANEL screen operations are implemented on the Amiga by opening a custom screen and a full-screen window with no sizing or close gadgets. Various screen modes can be selected using *pnlmode* as described below.

The PANEL screen and window can be accessed through the global pointers **PASCRN** and **PAWIND**, and the TextFont structure is also available through the **PAFONT** pointer.

By default, no title bar is shown on the screen. This allows the full area of the window to be used for display. The absence of the title bar prevents you from pulling down the screen, so we have made a way for the user to turn on the title bar by means of a pull-down menu. Whenever a program is looking for keyboard input it will also look for the 'menu' (right) mouse button being used to drag either to 'show' or to 'hide' the title bar. Once the title bar is visible you can use the front/back gadgets in the normal way or use the title bar to drag down the whole screen. If you try to use the pull-down menu when your program is not checking for keyboard input, both menu check marks will disappear. You can of course switch to and from the Workbench screen using the Amiga-N and Amiga-M keys, even when the title bar is not shown. If a program under development appears to hang, always try using the Amiga-N key to see if there are any requesters on the workbench screen.

Input from the keyboard is processed though a *raw* console device, which returns escape sequences for the function keys (not raw keycodes). These escape sequences are analysed in exactly the same way as for other versions of PANEL, using prefix and ignore specifiers. For details, see the main PANEL reference manual. The keyboard input process also checks for menu selections as described above.

To check for other input events, assign a value to **PAIDCM**, which is a pointer to a fucntion returning an integer. Your function will be passed the IDCMP message.

Output to the PANEL window uses the Amiga graphics text functions for maximum speed. Saving and restoring screen contents for field overlays is done with BltBitMap, and appears instantaneous. All output appears in one backdrop window, and the Amiga layer library is not used. The DOS cursor is not used, and PANEL implements a cursor as a graphics rectangle, also with BltBitMap. Box/Borders are implemented as a series of line-draws. The only visible change to PANEL supplied source code is that in the Amiga version the field save pointers **PAFSVA** are BitMap structures instead of the character pointers in other versions. They are also defined in the low-level module instead of in the validation source, so that **PAOSTERM** can clean up any that you forgot to 'unshow' or 'lose'.

The provided source code in PNL1C1xC.C allows all aspects of the PANEL Intuition interface and keyboard input environment to be modified as required. To make more extensive changes may require our 'full library source' option.

Icons are provided in *.info* files for all the PANEL programs except **pnlindx** (which always requires a command tail). Each program can also be called from the CLI, using program arguments if applicable. The **testc** program has been modified for the Amiga to require a carriage return after printing verification of the entries to stdout.

The PCMU ColorMap Utility

pcmu is the PANEL ColorMap Utility. It is used to maintain the file *Panel:pnlcolors* which contains eight sets of three numeric values for the Red, Green, and Blue settings of the corresponding Amiga colour register.

The *pnlcolors* file is listable. Each line in the file must be exactly bytes long, plus a newline, and must contain three values between 0 and 15 inclusive. When the 'save' option is selected, a backup copy of the file is made as *PANEL:pnlcolors.bak*.

To run **PCMU**, enter the name on a CLI command line or select the **pcmu** icon. Adjust the R, G, or B values by clicking the select button on the appropriate numeric value in the three columns on the right of the screen, and then clicking one or more times on the '++' or '--' as required. The default colours for PANEL and for the **pcmu** screen are foreground of 1 and background of zero. If you adjust these two colours to the same register values you will have to select your mouse click targets by feel rather than by sight.

When you have set all the colours you wish to use, make a note of up to sixteen pairs of foreground and background colour numbers (shown in the test colour patches) each PANEL highlighting type which you plan to use. We strongly recommend that the highlighting type zero (default) should use foreground colour 1 and background colour 0. These pairs of values are then set using *paiatt* as described in the next section.

Highlighting Attributes

The file *Panel:paiatt* contains pairs of foreground-background colour numbers for each of the sixteen PANEL highlighting types. The (up to 32) numeric digits are written in the file with no embedded spaces. If you specify fewer than sixteen pairs, or if the file is absent, the defaults from **PAIATT** in PNL1C1xC.C are used.

The default attribute values are as follows:

```
1 0  2 5  1 3  4 2  6 3  1 0  1 0  1 0   /* 0 .. 7 */
1 0  1 0  1 0  1 0  1 0  1 0  1 0  0 0   /* 8 .. 15 */
```

The defaults are selected to correspond roughly to the PANEL IBM-PC colour defaults when the supplied *pnlcolors* file is in use.

Note that text written with the same foreground and background colour numbers will be invisible unless a highlight modifier is used.

In this implementation, the highlight modifier byte high nibble is or'ed with the foreground colour and the low nibble or'ed with the backgound colour. As an example, a highlight modifier of hexadecimal 54 used with highlight type 15 will give foreground colour 5 and background colour 4. The actual colours will depend

on the *pnlcolors* settings. (Highlight modifers are specified as decimal values in the **panel** program 'Define Attributes' submenu).

We recommend that the *paiatt* file should start "10...." so that the default background is colour zero and the foreground is colour 1.

Keyboard Functions

The default keyboard functions are assigned as follows:

Carriage Return	Return
Escape	Escape
Destructive Backspace	Back Space
Cursor Left	Left Arrow
Cursor Right	Right Arrow
Cursor Up	Up Arrow
Cursor Down	Down Arrow
Erase Line	F9
Insert Character	F6
Delete Character	DEL
Insert Line	F7
Delete Line	F8
Tabulate	TAB
Query	HELP
Home	F10
Program Function 1-5	F1 - F5

These functions can only be changed by modifying the **PAIKEY** array in PNL1C1xC.C. The **PAIIGN** array may also need to be changed: it contains a '1' for every entry in *PAIKEY* which has a trailing character to be ignored.

Screen Mode

The file *Panel:pnlmode*, if present, contains codes to override the default mode. The mode can also be overridden by placing a value in **PAMODE** before calling **PAOSINT**. Defines for the **PAMODE** settings are supplied in PNLI.H.

Entries in the *pnlmode* file consist of upper-case letters or numbers as follows:

S	(PANELSIXTY) Set font size to sixty characters per line.
L	(PANELLORES) Set low resolution mode.
I	(PANELINTERLACE) Set Interlace mode.
Q	(PANELQUIET) Do not use narrator device.
1-5	Set screen depth to the given value.

We recommend experimentation with some of the modes before deciding to use them in an application. If you use the 60-column or low resolution modes you should ensure that you do not have messages or screen fields which overlap the screen as this can sometimes crash the system. This generally prevents the **panel** and **pantest** programs being used other than in 80-column high-resolution mode.

Global Variables

The following global variables are specific to the Amiga version of PANEL. If you use them in your program, it should normally be done inside an '#ifdef AMIGA' block to ensure portability.

PAMODE	Set before calling **PAOSINIT**, with the flags defined in PNLI.H or'ed with each other and with the depth value.
PATBSH	Set before calling **PAOSINIT**, with a value of TRUE to force initial showing of Screen Title Bar. During execution, has current status of title bar (TRUE if shown).
PADEPT	Screen depth (default 3). Set by **PAOSINIT**.

PACOLS Number of columns (default 80), set by **PAOSINIT.**

PAROWS Number of rows (default 25), set by **PAOSINIT.**

PANROK Narrator OK to use (default TRUE), set by **PAOSINIT.**

PACWID Character width in pixels (default 8), set by **PAOSINIT.**

PACHEI Character height in pixels (default 8), set by **PAOSINIT.**

PAADAT AmigaDOS date (array of 3 ints), set by **PAOSINIT.**

PASCRN Pointer to PANEL Screen structure.

PAWIND Pointer to PANEL Window structure.

PAFONT Pointer to PANEL TextFont structure.

PANARB PANEL Narrator IO Request Block.

PAIDCM Pointer to user function to process IDCMP messages.

Speech Functions

The PANEL Amiga version includes an additional Extended Attribute specifier, *HS*, for spoken help messages. The extended attribute string consists of a single string to be passed to the translator and narrator device.

The PANEL speech function is implemented by the function **PANARR**, for which the source is provided in PNL1C1xC.C. The passed string can contain escape sequences introduced by a vertical bar character followed by the single character 'M', 'F', 'N', or 'R' (upper or lower case) for Male, Female, Natural, or Robotic respectively. The default is Male, Natural.

You can modify the PANEL validation routines to allow spoken error messages as well as the help messages provided, and **PANARR** can also be called by your application program.

pangenc

lease note the description of the **pangenc** option '-s'. This switch is pre-set for the cross-development version of PANEL in the PANGENC.MSG file which is deleivered on the PANEL Amiga library diskette. Do not use the PANGENC.MSG file on the 'common C' diskette.

Problems

1. We have 'commented out' the word 'register' in the Lattice version of the PNLI.H header file. Although Lattice have fixed earlier problems with register variables, some of the PANEL functions failed to work when we used register variables. The code generation by Lattice looks good, and problems seem to occur when an Amiga function is called during a loop using register variables: however Commodore deny failing to restore register values after system calls.

2. In creating a C-Tree library for use with the **pd** program we found that it was necessary to use the define option *FPUTONLY* in CTOPTN.H to avoid problems caused by seeking beyond end-of-file. This slows down the addition of records to a file, but at present there seems to be no alternative.

3. We have not tested output to the printer from **pantest** or **pd**.

4. It is particularly important to call **PAOSINIT** and **PAOSTERM** at the start and end respectively of every program using PANEL functions. On an MS-DOS version of PANEL, these routines did nothing, so you may have failed to call them without ill effects. Also, failing to call **PAINIT** will not pass unnoticed in this version.

Section VI
Index

Index

68000	136
Abbreviated fcb	67, 87, 131
Additional keys	42
Alphabets	51
Amiga	208
Amiga cross-compiler	210
Amigados	208
ANSI console driver	9, 10
Answer function (PAANSW)	63
Apricot	129
Attribute bytes	117, 145
Attribute substrings	56
Attributes	51
Attributes, extended	56
Background colour	64
Backspace cursor	115
BASIC	192
Basic generated code	196
Basic sample files	205
BD extended attribute	53
Beep	40
Bell	78, 115
BIX	3
BLOCK DATA subprogram	187, 188
Block mode terminal	43
Border characters	118
Border colour	64
Border legend	54
Borders	53
Box borders	53, 212
Boxes	41
Brackets, in message	94
Buffered screen output	65, 146
Byte Information Exchange	3
C calls from FORTRAN	177
C-Tree	149, 217
Carriage return key	36
CEXEC Microsoft library	173, 187
Change highlighting	105
Character list	57
Character output	146
Character validation	24, 70
CL Extended Attribute	57
Clear screen	78, 114
Cli (Amiga)	212
Code generation	134
Coloured fields	44
COMMAND.COM	192, 195
Comments, PANEL.VDU	124
Commodore Amiga	208
Compaq	11, 127

Compiler selection 155
Compiler warnings 45
Computer Innovations C86 135
Concatenate panel files 153
CONFIG.SYS 10
Console status 68
CPAVDU 83
cpio 7
CR required attribute 51
Currency attribute 52
Currency field 60
Current date 165
Current field (PACRF) 67
Cursor coordinates 41
Cursor keys 37
Cursor positioning 69, 119, 142
Custom numeric format 61

Data management 6
Data management program 51, 149
datablock 22, 132
dataheight 132
datawidth 132
Date validation 165
Date, current 165
Decimal places 61
Delays 115
Delete character key 38
Delete line key 38
Destructive backspace key 37
Display character 66
Display field 102
Displayable characters 72
Dynamic load 15, 45, 87

Echo input attribute 51
enum variable 135
Environment control block 72
Environment variables 7, 55, 83
Erase end-of-line 78, 114
Erase line key 37
Escape key 36, 84, 116
Escape sequence output 146
Escape sequences 158
Exit (DOS command) 195
Exponent format 62
Extended attributes 17, 56, 101
Extended PF key table 124

Faircom 149
field control block (FORTRAN) 176
Field control blocks 21, 67, 131
Field data areas 22, 106
Field display 102
Field index 74

Index

Field name 100
Field overlay 102, 131
Field pointer table 76
Field read 104
Field save pointer 102
Field validation 24, 79
Field-level function 102
Fill character 98
Floating point 45
FORTRAN 172
FORTRAN calls to C 177
FORTRAN validation 188
Free panel space 77
Full fcb 67, 87, 131
Function keys 42

General validation 24, 148

HB Extended Attribute 58
Header files (.h) 155
Help box 14, 58
Help fields 44
Help key 39
Help message 14, 56
Help messages 216
Hewlett Packard 150 41, 129, 203
High bit in data 136
Highlight method 10, 11, 81, 117
Highlight modifier 10, 11, 81
Highlight sequences 118
Highlight types 11
Highlighting 66, 81
Highlighting (Amiga) 213
HM Extended Attribute 56
Home key 39

I library 13
IBM Basic 192
IBM PC 40, 64, 122, 124, 194
IBM PC colour display page 11, 141
IBM PC display methods 9
IBM PC screen attributes 125
Ignored input characters 123
Index to multiple panels 96, 153
Initialise screen 78
Input functions 123
Insert character key 38
Insert line key 38
Integer field 60
Interlace mode (Amiga) 215
Intuition 210

Key field attribute 51
Key fields 149
Key functions 80, 214

Key validation 24, 85
Keyboard codes 121
Keyboard delay 116
keyboard program 59
Keyboard read 65
Keyboard specification 159
Keyboard status 116

Legend, in border 54
Levels 27, 97, 100
Line operation 60, 89, 106
Line validation 24, 89, 106
Line-draw characters 53
Linking 'I' library 13
Linking FORTRAN programs 187
Linking programs 23
Lo-res mode (Amiga) 215
LOC function (FORTRAN) 177
Lose overlay field 103
Low-level I/O module 12, 26

Mask entry 15, 100
Matrix of fields 44
Memory-mapped I/O 64, 141
Menu field 15
Message field control block 63, 93
Message heading 93
Message line 93, 100
Message line clear 94
Message line repeat 94
Message processing 93
Microsoft Basic 192
Microsoft FORTRAN 172
Microsoft Windows 10
Module names 20
Module numbers 20
MSG environment variable 55
Multiple panel file 96, 139, 153

National alphabets 51
Native Amiga compiler 210
Negative highlight type 54
NF Extended Attribute 60, 61
No input attribute 51
NSAVES 29
Numeric attribute 52
Numeric field custom format 61
Numeric fields 46, 60
Numeric range check 61
Numeric validation 60, 163

Operating system init. 140
Overlapping fields 97
Overlay field 27, 31, 102, 131
Overlay Mode 28

Index

Overlay threshhold 28

PAADAT (Amiga date) 216
PAANSW (answer function) 63, 93
PAANSW (Basic) 202
PAANSW (FORTRAN) 178
PABF (block fill) 25
PABF (FORTRAN) 186
PABM (block move) 25
PABM (FORTRAN) 186
PACHCL (change colour) 11, 64
PACHCL (FORTRAN) 180
PACHEI (Amiga char height) 216
PACINP (char. input) 65
PACINP (FORTRAN) 179
PACOLS (Amiga columns) 216
PACOUT (char. output) 66
PACOUT (FORTRAN) 179
PACRF (current field) 67
PACSTA (console status) 68
PACSTA (FORTRAN) 179
PACURS (cursor positioning) 69
PACURS (FORTRAN) 180
PACVAL (char. validation) 24, 70
PACWID (Amiga char width) 216
PADEPT (Amiga depth) 215
PADSP (displayable chars.) 72
PAECB (Basic) 203
PAECB (env. control block) 72, 84, 143
PAEOUT (escape output) 146
PAEOUT (FORTRAN) 179
PAFLX (field index) 43, 74, 79, 109
PAFLXS (saved field index) 43, 74, 110
PAFONT (Amiga font) 210, 216
PAFPT (field pointer table) 21, 22, 76, 109, 143, 147
PAFREE (FORTRAN) 182
PAFREE (free loaded panel) 77
PAFSVA (Amiga BitMap) 212
PAFSVA (field save areas) 102, 107, 131
PAFUNC (FORTRAN) 178
PAFUNC (screen function) 21, 78, 109
PAFVAL (field validation) 24, 79
PAGETF (FORTRAN) 179
PAGETF (get key function) 80, 85
PAHIOF (FORTRAN) 180
PAHIOF (highlight off) 81
PAHION (FORTRAN) 180
PAHION (highlight on) 81
PAHTYP (FORTRAN) 180
PAHTYP (highlight type) 81
paiatt (Amiga) 209
PAIATT (attribute array) 12
PAIATT environment var. 12
PAIATT environment variable 55
PAIBOR environment var. 12

PAIBOR environment variable 55
PAICAT (colour attributes) 12
PAIDCM (Amiga IDCMP) 216
PAIDEN (version identifier) 72
PAIIGN (ignore table) 12
PAIKEY (Amiga) 209
PAIKEY (keyboard table) 12
PAIMAT (mono attributes) 12
PAINIT (FORTRAN) 177
PAINIT (PANEL initialise) 21, 72, 83
PAIX (string index) 25
PAKTS (keyboard table size) 12, 42, 84, 124
PAKVAL (keyboard validation) 24, 42, 85, 168
PALIFT (FORTRAN) 181
PALIFT (lift screen area) 144
PALOAD (dynamic panel load) 77, 87, 138
PALOAD (FORTRAN) 182
PALOAD (load panel) 21, 22
PALOOK (FORTRAN) 182
PALOOK (search in index) 87, 96
PALVAL (line validation) 24, 89
PAMEN (message end point) 63, 93
PAMENU (Basic) 205
PAMENU (FORTRAN) 181
PAMENU (pop-up menu function) 30, 91
PAMENU (pop-up menu) 25
PAMFCB (message fcb) 63, 93
PAMHD (message heading) 93
PAMLCL (FORTRAN) 182
PAMLCL (multiple file close) 96
PAMLN (Basic) 203
PAMLN (message line) 63, 93, 147
PAMLOP (FORTRAN) 182
PAMLOP (multiple file open) 95
PAMODE (Amiga mode) 215
PAMSGE (Basic) 201
PAMSGE (FORTRAN) 178
PAMSGE (message function) 93
PAMULT (multiple file switch) 87, 96
panafcb (abbreviated fcb) 131
PANARB (Amiga narrator rb) 216
panecb (env. control block) 72
panel commands 98
panel control block (FORTRAN) 176
Panel control block (pcb) 87, 138
Panel file field sequence 154
panel program 97
panel program functions 5
PANEL system components 4
PANEL.MSG 56
PANEL.VDU 5, 11, 59, 83, 112, 156, 167
PANEL.VDU structure 130
Panel: device (Amiga) 209
PANELF (Basic) 201
PANELF (field function) 21, 102

Index

PANELF (FORTRAN) 181
PANELF return codes 107
paneli program 13, 97
PANELS (Basic) 201
PANELS (FORTRAN) 181
PANELS (screen function) 21, 74, 79, 109
PANELS return codes 110
paneltc program 111
panfcb (full fcb) 131
PANFE (numeric field expand) 175
PANFL (number of fields) 76, 147
pangen04 program 187
pangenb program 192, 193
pangenc (Amiga) 208
pangenc progam 21
pangenc program 22, 134
pangenc program (Amiga) 217
pangenc.msg 134
PANLOW (Basic functions) 199
panpcb (panel control block) 138
PANROK (Amiga narrator) 216
pantest program 32, 139
pantesti program 13, 139
PANUMD (numeric array) 46, 61
PANUMI (numeric array) 61
PANUML (numeric array) 61
PANUMR (numeric array) 61
PANUMV (numeric validation) 25
PANUNL (numeric unload) 175
PAOLDH (old highlight type) 105
PAOSINIT (Amiga) 217
PAOSINIT (Op. sys. init.) 21, 140
PAOSTERM (Amiga) 212, 217
PAOSTERM (Op. sys. term.) 21, 140
PAOVTH (Basic) 204
PAOVTH (overlay threshhold) 28, 30, 31, 32, 44, 102
PAPAGE (colour display page) 11, 64, 141
PAPAGE (FORTRAN) 180
PAPSHI (Basic) 204
PAPSHI (PANELS highest level) 27, 109, 143
PAPSLO (Basic) 204
PAPSLO (PANELS lowest level) 27, 109, 143
PAPSLV (level determination) 25
PARDHI (Basic) 203
PARDHI (highlight read field) 16, 104
PARDSP (FORTRAN) 181
PARDSP (redisp. screen area) 144
PAROUT (FORTRAN) 179
PAROUT (raw output) 146
PAROWS (Amiga rows) 216
PARSTC (reset cursor) 25
PASBAD (FORTRAN) 180
PASBAD (screen write address) 142
PASCNL (Basic) 202
PASCNL (screen line contents) 143

PASCRN (Amiga screen) 210, 216
PASCRO (vertical scroll) 168
PASCSE (Basic) 205
PASCSE (FORTRAN) 178
PASCSE (screen save enable) 21, 28, 92, 144
PASERV (Basic service fns) 203
PASET (of fields for PANELS) 74, 109
PASOUT (FORTRAN) 181
PASOUT (string output) 146
Password validation 163
PATB (FORTRAN test bit) 189
PATBSH (Amiga title show) 215
PATDSP (test displayable) 25
Path names 55
PAUSEP (Basic) 197, 200, 201
PAUSEP (FORTRAN) 181
PAUSEP (use panel) 21, 76, 88, 147
PAVAL (valid characters) 57
PAWIND (Amiga window) 210, 216
PAXASC (extended attr. scan) 25
PAXVAL (general validation) 24, 57, 148
pcmu (Amiga ColorMap) 210, 212
PD environment variable 55, 149
pd program 149
PFALST (allocation setup) 174, 177
PFGET (FORTRAN variable) 182
PFGETD (FORTRAN data) 184
PFGETF (FORTRAN fcb) 185
PFGETI (FORTRAN variable) 184
PFPUT (FORTRAN variable) 183
PFPUTD (FORTRAN data) 185
PFPUTF (FORTRAN fcb) 185
PFPUTI (FORTRAN variable) 184
PFPUTP (FORTRAN pointer) 186, 188
PFSCNL (FORTRAN screen line) 182
PFSETP (FORTRAN setup) 174, 178
PFXCSN (FORTRAN xa scan) 185
Physical Attribute 28
Picture validation 15
Picture Validation 57
Picture validation 163
PNB files 154
PNL environment variable 55
PNL files 154
pnl3.h 73, 132, 155
pnl4.h 26, 71, 86, 90, 155
pnl5.h 75, 155
pnlbas program 192, 194
pnlbasi program 194
pnlc.h 155
pnlcolors (Amiga) 209, 212
pnli.h 155
pnlindx program 96, 139, 153
pnlindx program (Amiga) 212
pnlmode (Amiga) 209

Index

PNLMONO environment var. 12
Pop-up entry window 30
Pop-up fields 27
Pop-up menu 91, 164
Portability 43, 155
Prefix character 121
Program function keys 39
Prompt line 100
PSEG file 194
PSETUP (Basic setup) 199
PV Extended Attribute 57

Query key 39

Range check 61
ranlib 7
Read field 104
Recursive PANELF call 71, 86, 90, 107
register variables 217
Restore cursor position 71, 86, 90
Right justify attribute 52, 61
ROM BIOS calls 9, 10, 11, 64, 84, 141, 144

Screen border colour 64
Screen delays 115
Screen editor (panel) 97
Screen line contents 143
Screen mode (Amiga) 215
Screen output 146
Screen save 41, 66
Screen save enable 28, 144
Screen save pointers 29
Screen size 119
Screen writing address 142
Screen-level function 109
Set Graphics Rendition (SGR) 10
Sirius 1 128
Spoken help messages 216
sprintf function 60
sscanf function 60
startup-sequence (Amiga) 209
String output 146
Subroutine Library 20
Support 3

Tabulate key 38
TAILOR environment variable 55, 84
tailor program 10, 156
TAILOR.DAT 156, 162
tar (tape archive) format 7
TERM environment variable 55, 83, 111
termcap 5, 8, 111
TERMID (terminal ID) 72
Terminal ID 113
testc program 14

testci program 13
Texas Instruments PC 127
Title bar 210
Topview 10

Undefined attribute 52
Undefined symbols 45
Unshow 103, 131
Unshow field 29
Upper case attribute 51
Use panel (PAUSEP) 147
User attribute 52, 60
User interface 14, 35

Valid characters 163
Validation 24, 70, 79, 85, 89, 148, 163
Validation parameters 26, 71, 86, 90
Validation pointer table 24
Validaton (FORTRAN) 188
Values list 57, 164
VDU environment variable 55
VDU Initialisation 113
VDU Termination 114
vdugenc program 83, 167
Vertical menu 91
Vertical scroll 14, 90, 168
Victor 9000 128
Visual attributes 117
VL Extended Attribute 57

Wang PC 128
Warning messages 45
Windows (Microsoft) 10
Windows processing 144
Wipe attribute 52, 61
Word-wrap 15, 164

X attribute 52
Xenix 172

Zero values 46, 60

*****READ CAREFULLY BEFORE OPENING*****

BY OPENING THIS PACKAGE YOU INDICATE YOUR ACCEPTANCE OF THESE TERMS AND CONDITIONS.

PROGRAM LICENSE AND DISTRIBUTION AGREEMENT

1. This software product is licensed to you for your personal use on a single computer system at a time. You must return the enclosed registration card to Lattice in order to identify yourself as the Licensee.

2. You may make backup copies of the program diskette(s) and may also copy the software to another storage medium for your personal use. You must remove all copies from the machine that you are no longer using.

3. You may not give copies of the product to another person, transfer the license to another person, or share the use of the product with others (e.g. via a network) without a specific agreement with Lattice.

4. Lattice reserves the right to terminate this license upon breach. You may terminate the license at any time by destroying all copies of the product. You should also notify Lattice at that time so that you will be removed from the software update notification list.

5. You may distribute derivative works (i.e. programs generated by and/or including binary code parts of the Lattice product in such a way that they cannot be separated out) as you desire with no further fees paid to Lattice. Source code of the product may NOT be distributed without a specific agreement with Lattice.

6. Product specifications and features are subject to change without notice.

7. This agreement may be superseded by a direct agreement with Lattice.

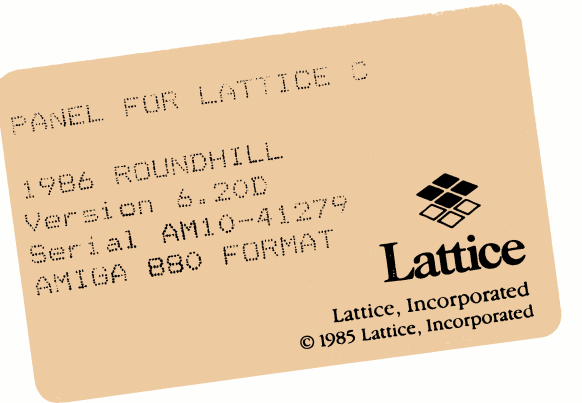
8. The enclosed Master Diskette(s) are your proof of purchase and must be returned to Lattice in order to receive product updates or upgrades when they are made available. However, if Lattice has not received your registration card, Lattice is under no obligation to make availiable to you any updates or upgrades even though you may have made payments of any applicable fee.

LIMITED WARRANTY

1. Lattice warrants the diskette(s) on which the program is furnished to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of delivery to you as evidenced by a copy of your sales receipt.

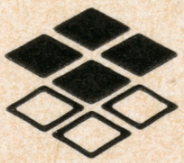
2. Lattice does NOT warrant that the program will meet your requirements or that the operation of the program will be uninterrupted and error free. You are solely responsible for the selection of the program to achieve your intended results and for the results actually obtained.

THIS AGREEMENT WILL BE GOVERNED BY THE LAWS OF THE STATE OF ILLINOIS


PANEL FOR LATTICE C
1986 ROUNDHILL
Version 6.20D
Serial AM10-41279
AMIGA 880 FORMAT
Lattice
Lattice, Incorporated
© 1985 Lattice, Incorporated

PANEL FOR LATTICE C
1986 ROUNDHILL

Version 6.20D
Serial
AMIGA 880K FORMAT
Disk 1 of 2



Lattice

Lattice, Incorporated
© 1985 Lattice, Incorporated

PANEL FOR LATTICE C

1986 ROUNDHILL
Version 6.20D
Serial
AMIGA 880K FORMAT
Disk 2 of 2

Lattice

Lattice, Incorporated
© 1985 Lattice, Incorporated

PANEL SYSTEM: SOFTWARE LICENCE AGREEMENT

IF YOU OPEN THE SEALED PACKAGE CONTAINING THE PANEL SYSTEM MATERIALS WE SHALL ASSUME THAT YOU AGREE TO THE TERMS OF THIS AGREEMENT. RETURN THE PACKAGE UNOPENED IF YOU CANNOT ACCEPT THESE TERMS.

WE are Roundhill Computer Systems Limited, of P.O. Box 14, Marlborough, Wiltshire, SN8 1LG, England, originators of the PANEL system.

YOU are the purchaser of the PANEL system, which includes, for a company, corporation or educational body, your employees or any other person authorised to use the product.

1. We license you to use the PANEL system on a single computer at a single location. You agree to purchase additional copies of the PANEL system if you wish to use it on more than one computer. This licence will continue for as long as you keep to the terms of this agreement. If we terminate your licence you agree to destroy all copies of the PANEL system, or any part of the PANEL system, in your possession.

2. You may not transfer the PANEL system or any part of it to any other party, except as provided by clause 7 below, without first informing us and forwarding to us the other party's written consent to these terms. You will then destroy any copies of the PANEL system, or any part of the PANEL system, in your possession.

3. We believe that the PANEL system will perform as stated in the PANEL reference manuals. However we cannot accept any responsibility or liability if it fails to do so. If you notify us of an error, or if we find one, we shall use our best endeavours to correct in the next release of the product. If you have returned the registration card, you will be eligible to purchase an upgrade, if we issue one, to the current release at the then current fee.

4. Because we cannot know what you expected of the PANEL system when you purchased it, we have to say that we cannot actually guarantee that PANEL will do anything you may have read or heard that it will do, nor can we be liable for any consequences whatsoever of your purchase of it.

5. You agree to return the registration card before making any use of PANEL. However your failure to return the card will not alter your implied agreement to these terms by opening the sealed package.

6. You will make no more than three copies of the PANEL system, solely for backup purposes, and will label any copy with our copyright notice, as on the original. You will not copy any of the documentation. Further copies of the documentation may be purchased.

7. Subroutine libraries included as part of the PANEL system are designed to be linked with your program modules to generate composite programs. We permit you to distribute such composite programs, and the PNLBAS module from the PANEL Basic Programming Language Interface, without payment of any further fee, provided that you place a notice 'Portions (C) Roundhill Computer Systems Limited' on the media label. If you wish to distribute the TAILOR program with your composite programs, we will offer you a separate licence agreement covering the supply of multiple copies of TAILOR and its documentation.

8. You acknowledge that we will vigorously pursue a claim for damages against you should any loss result from the failure by you to comply with these terms.

9. This agreement is governed by the laws of England.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO. 704 GLEN ELLYN, ILLINOIS 60138

POSTAGE WILL BE PAID BY ADDRESSEE

Lattice, Incorporated
P.O. Box 3072
Glen Ellyn, IL 60138

Lattice

SecretDisk

When your business is nobody else's.

SecretDisk™

Now you can keep your confidential data confidential. Thanks to new SecretDisk from Lattice. SecretDisk will protect your programs and data from unauthorized access, and won't annoy you in the process. When starting your computer, enter your password and your secured files will behave just like normal files. During the day you can switch access off instantly or regain access with a password at any time.

SecretDisk works with all your other software. And for just $59.95 you'll come away with peace of mind knowing that an intruder won't come away with a piece of information.

So if you have data that's for your eyes only—like payroll, billing, stock information or anything else that you consider to be nobody else's business, SecretDisk will make sure that the information on your personal computer remains personal.

- **Fast, less than 1% overhead**
- **No hassle** • **No hardware to install**
- **No copy protection**

Only $59.95

Lattice

Lattice, Incorporated
Post Office Box 3072
Glen Ellyn, Illinois 60138

To order, send in the card below:

Yes, I agree that my business is nobody else's.

Send me _____ SecretDisk/s @ $59.95 each. I understand that if I am not satisfied that I can return the product within 30 days for a *full* refund.

Name _____

Company Name _____

Street Address _____ City _____ State, Zip _____

Form of payment: ☐ Master Card ☐ Visa ☐ AX ☐ Check

Name on Charge Card _____

Card Number _____ Exp. Date _____

Signature _____

Illinois residents add 6% sales tax.

Notice: This Product is not allowed to be exported outside of the U.S.A. due to its sophisticated encryption algorithms.

Lattice

To All Registered Lattice Users

We won't keep it a secret...

the price of our SecretDisk™ data encryption software is going up to $119.95 on July 1.

The enclosed postcard, sent to Business Week readers, represents a limited time offer for SecretDisk at an introductory price. As a registered Lattice user, we want to make you the same offer: SecretDisk at $59.95.

If your business is nobody else's, return the enclosed card to us before July 1, and beat the price increase!

SAH

Steve Hersee

P.S. For more info on SecretDisk, see the review in the February 18th issue of PC WEEK.

Lattice, Incorporated
P.O. Box 3072, Glen Ellyn, IL 60138 • 312/858-7950 • TWX 910-291-2190

Lattice

Take this simple test to reveal the best tools for you.

1. Which company provides more C language compilers, libraries and utilities than any other?
2. Which C compiler has more third-party library, utilities and debugger support as well as more support from independent bulletin boards?
3. Which company gives you an unconditional money-back guarantee on all software products?
4. Which C compiler is used by more professional software developers than any other?
5. Which company provides a technical hotline that lets you speak to programmers who can help you use the products more effectively?

Answers:

1. Lattice, Incorporated
2. The Lattice C Compiler
3. Lattice, Incorporated
4. The Lattice C Compiler
5. Lattice, Incorporated

Lattice products help you create better programs!

Lattice, Incorporated delivers the professional tools a programmer needs.

Lattice C Compiler $500.00

With more than 30,000 users worldwide, Lattice C Compilers set the industry standard for MS-DOS and PC-DOS computers. Lattice C gives you all you need for development of programs on the Intel iAPX86 processor family. Lattice C is a full implementation of Kernighan and Ritchie with the ANSI C extensions and many additional features for MS-DOS and PC-DOS.

C-SPRITE™ $175.00

Simplify debugging of programs written in Lattice C and most other languages. C-SPRITE is the first software tool for *both* source level and machine level debugging. It gives you a high-level view of C programs with function names, source lines, variable names and C data types. In addition, C-SPRITE provides a low-level view of machine addresses and instructions. Debugging through a communication port, help screens, macros, command files, conditional commands and support for PLINK86 overlays are also provided.

Lattice Screen Editor™ *(LSE)* $125.00

Designed as a programmer's editor, LSE is fast, flexible and easy to learn. LSE's multi-window environment provides all the editor functions you need including block moves, pattern searches and "cut and paste." In addition, LSE offers special features for programmers such as an error tracking mode and three Assembly Language input modes. You can also create macros or customize keystrokes, menus, and prompts to your style and preferences.

dBC III™ $250.00

The dBC III library lets you create, access and update files that are compatible with Ashton-Tate's dBASE system. dBC III's C functions let you extend existing dBASE applications or allow your users to process their data using dBC III or dBASE III.

LMK™ $195.00

An automated product generation utility compatible with UNIX Make, LMK lets you rebuild complex programs with a single command. Once you specify the relationships of the various pieces of your system in a dependency file, LMK automatically rebuilds your system the same way every time, and only compiles program files that have changed. But LMK is not limited to updating programs. You can use LMK to update documentation or perform **any** executable command!

Curses™ $125.00

The Lattice Curses screen interface is a library of 84 C functions used for updating the screen of an IBM PC. It is compatible with UNIX curses packages, and can be used to transfer UNIX curses programs to the PC. Lattice Curses can keep and update any number of full or partial virtual screen images in memory and paint the actual screen as needed with one or more of the images. Functions are provided to write text, move cursors, scroll, overlay, outline, insert, delete, clear and highlight the virtual screens.

db_VISTA™ $395.00

db_VISTA will reduce your program development time for large application systems. Designed for C language applications programming, db_VISTA consists of a Data Definition Language (DDL) processor and library of C functions. You specify the structure of the database in the DDL. The DDL processor then compiles the specifications into a set of tables to be used by the db_VISTA library functions which you call to manipulate and access the database.

Text Management Utilities™ *(TMU)* $120.00

TMU consists of eight software tools to help you manage your text files. **GREP** searches files for the specified pattern. **DIFF** compares two files and lists their differences. **EXTRACT** creates a list of file names to be extracted from the current directory. **BUILD** creates batch files from a previously generated filename list. **WC** displays the number of characters and optionally the checksum of a specified file. **ED** is a line editor which can utilize output from other TMU software in an automated batch mode. **SPLAT** searches files for a specified character string and replaces every occurrence with a specified string. And **FILES** lists, copies, erases or removes files or entire directory structures which meet the specified conditions.

TMU is not system or language oriented. You can use it on files created with a text editor or produced by a text processor. Files written in C, Assembler, COBOL, Fortran, Pascal or English can all be handled by TMU.

Panel™ $295.00

Panel will help you write your screen programs and layer your screen designs with up to ten overlapping images. Panel's screen layouts can be assigned to individual windows and may be dynamically loaded from files or compiled into a program. Panel will output C source for including in your applications. A monitor and keyboard utility is also included to allow you to customize your applications for other systems.

SecretDisk™ $59.95

SecretDisk gives you automatic data encryption for your sensitive files. SecretDisk is loaded as a disk driver by MS-DOS. It creates new "SecretDisk drives" on your floppy or hard disk where *all* data and programs are *always* fully encrypted. SecretDisk is easy to use. You get the convenience of a password system with the full protection of the NBS Data Encryption Standard. You enter your password when the system is booted, and switch protection on and off with a single password-controlled command line.

Lattice

SOFTWARE ORDER FORM

	Unit Price	Qty	Amount
LATTICE MS-DOS C COMPILER	$500	___	_____
C FUNCTION LIBRARIES			
C-Food Smorgasbord	$150	___	_____
dBC II Indexed File System (dBASE II Compatible)	250	___	_____
dBC III Indexed File System (dBASE III Compatible)	250	___	_____
Window Manager	295	___	_____
db_VISTA Database Library	395	___	_____
Curses Unix Compatible Screen Manager	125	___	_____
PANEL Forms Manager	295	___	_____
TopView Toolbasket	250	___	_____
C UTILITY PROGRAMS			
LSE Lattice Screen Editor	$125	___	_____
CVUE Screen Editor	75	___	_____
Text Management Utilities (GREP/DIFF/ED/WC/EXTRACT/BUILD)	120	___	_____
LMK (Like Unix "make" Utility)	195	___	_____
C Cross Reference Generator	50	___	_____
PLINK86 Overlay Linker	395	___	_____
Pre-C (Like Unix "lint" Utility)	395	___	_____
C-SPRITE Program Debugger	175	___	_____
SecretDisk	$59.95	___	_____
Total		___	$_____

WARRANTY

Lattice provides free updates for all products with Lattice serial numbers during the 120-day warranty period. All new updates require the return of the original diskettes to Lattice. Please keep them in a safe place. Lattice also provides an unconditional 30-day money-back guarantee.

Please send information on:

☐ C Cross Compilers ☐ Compiler Trade-Ins ☐ Corporate Licenses ☐ Lattice C Users Group ☐ Amiga Products ☐ Source Code

Return this order form today. Or call Lattice, Incorporated at (312) 858-7950 TWX 910-291-2190

CUSTOMER INFORMATION

Name:_____ Date:_____

Company Name:_____ Telephone: ( )_____-_____

Bill To:_____ Ship To:_____

_____ _____

_____ _____

_____ _____

Form of Payment:

☐ COD ☐ Master Card ☐ Visa ☐ American Express ☐ Check ☐ Purchase Order Enclosed

Card #_____ Exp. Date_____

Signature:_____

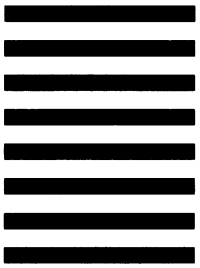
LID85

To Order: Detach order form and re-fold so the business reply postage is on the outside, tape closed, and drop in the mail.

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 704 GLEN ELLYN, IL 60138

POSTAGE WILL BE PAID BY ADDRESSEE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



Lattice, Incorporated
P.O. Box 3072
Glen Ellyn, IL 60138

PROGRAMS FOR PROGRAMMERS

For more than three years, Lattice has been providing a complete range of top quality programming tools for software developers. Lattice products are used by more than 30,000 software developers including companies such as IBM, Ashton-Tate, Microsoft and Lotus.

Lattice provides updates and enhancements to its products on a regular basis, and complete support is only a phone call away. Contact the Lattice Sales Department at (312) 858-7950 for details on any of the products listed. There is a very good chance that we have the exact programming tool you've been looking for.

Lattice, Incorporated
Post Office Box 3072
Glen Ellyn, Illinois 60138

FIRST-CLASS MAIL
U.S. POSTAGE
PAID
Glen Ellyn, IL
Permit No. 597

Simple test reveals best tools for you.

Special Offer to Lattice Customers

Four Products Only $699.00!

Save up to $296.00 off the individual purchase price when you obtain a complete set of programmer tools from Lattice. Take your choice of packages containing tools every C programmer needs, and take advantage of substantial savings! Call (312) 858-7950 or return the attached order form today! This is a limited-time special offer to current Lattice customers.

You Know You Can Trust Lattice Products

Lattice products are used by more than 30,000 software developers worldwide in companies such as IBM, Ashton-Tate, SAS Institute, MicroPro and Lotus. Lattice provides updates and enhancements on a regular basis with complete on-line support through two bulletin board services. And all Lattice products give you an unconditional 30-day money-back guarantee!

Lattice

Lattice, Incorporated
P.O. Box 3072
Glen Ellyn, IL 60138
Phone: (312) 858-7950
TWX: 910-291-2190
Telex: 532253

Lattice Products for the Amiga

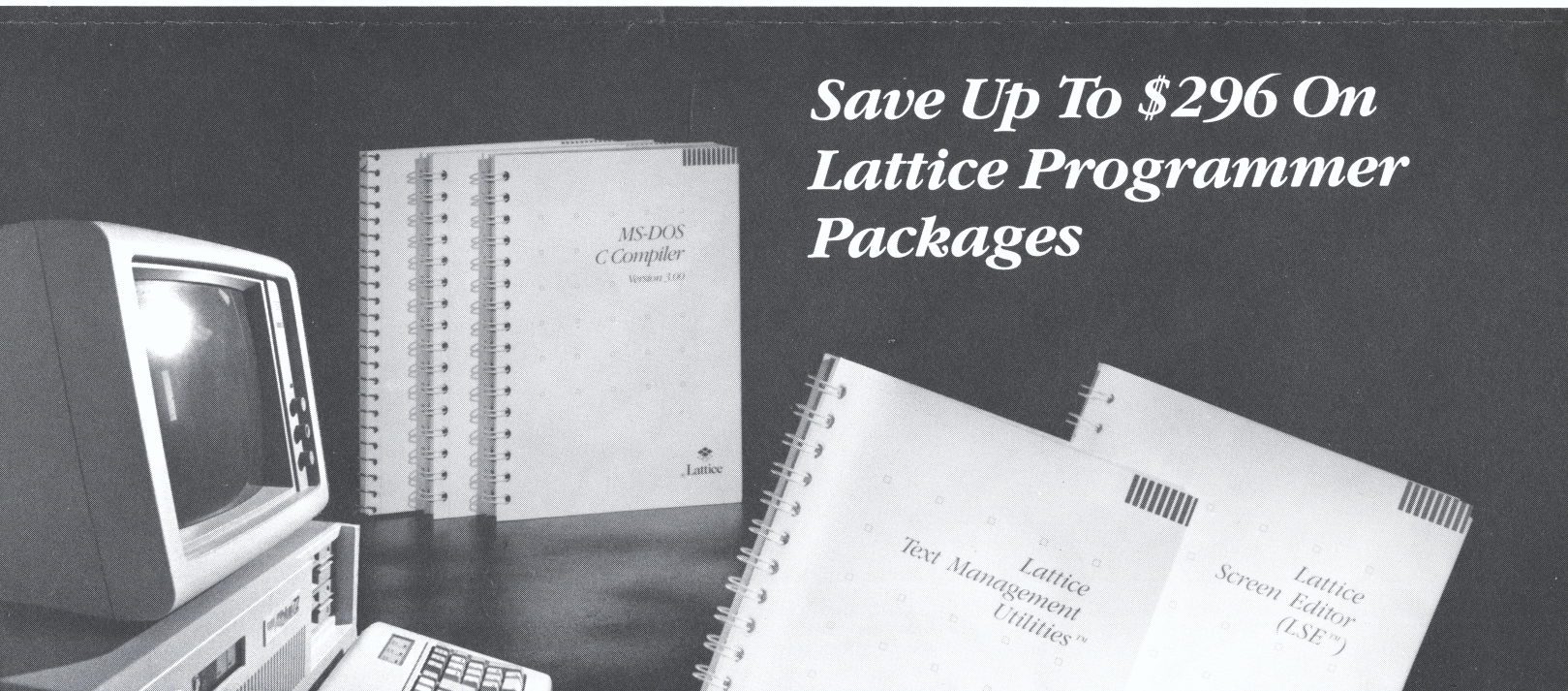
Lattice AmigaDOS C Compiler
Lattice MacLibrary
Lattice Make Utility
Lattice dBC III Library
Unicalc Electronic Spreadsheet
Lattice MS-DOS to AmigaDOS C Cross Compiler

Lattice Text Utilities
Lattice Screen Editor
Panel

Other Software Products From Lattice

Unicalc Electronic Spreadsheet
C-FOOD Smorgasbord Library
dBC II ISAM Library
CVUE Full Screen Editor
Lattice VAX/VMS to iAPX86 C Cross Compiler
Lattice SUN/UNIX to iAPX86 C Cross Compiler
Lattice MS-DOS to Z80 C Cross Compiler
Lattice VAX/UNIX to iAPX86 C Cross Compiler
LSE Lattice Screen Editor
db_VISTA
FirstTime
Instant-C
VHS Video C Training

PLINK86 Linker
Curses Screen Package
dBC III ISAM Library
Panel
SideTalk
SUN PLINK86
VAX PLINK86
CXREF
TopView Toolbasket
Pre-C
SecretDisk



Save Up To $296 On Lattice Programmer Packages

ALL THE TOOLS A PROGRAMMER NEEDS IN ONE PACKAGE

Both Package I and Package II include:

Lattice MS-DOS C Compiler Version 3.0

The industry standard for MS-DOS and PC-DOS computers, Lattice C gives you all you need for development of programs on the Intel iAPX86 processor family. Version 3.0 is a major upgrade and includes full implementation of Kernighan and Ritchie plus ANSI C extensions and many additional features to aid your MS-DOS and PC-DOS programming.

C-SPRITE Symbolic Debugger

C-SPRITE is the first software tool you can use for both source level and machine level debugging. You get a high-level view of your C programs with function names, source lines, variable names and C data types. In addition, C-SPRITE provides a low-level view of machine addresses and instructions. C-SPRITE also gives you debugging through a communications port, help screens, macros, command files, conditional commands and support for PLINK86 overlays.

LMK Automated Product Generation Utility

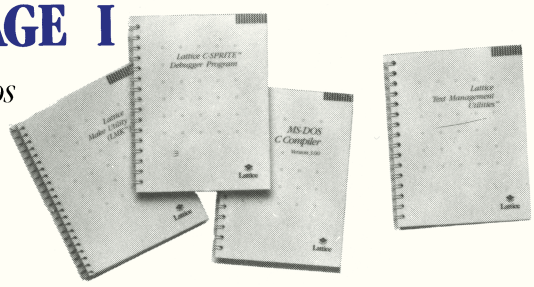
Like UNIX Make, LMK lets you rebuild complex programs with a single command. Once you specify the relationships of the pieces of your system in a dependency file, LMK automatically rebuilds your system the same way every time and only compiles program files that have changed. You can also use LMK to update documentation or perform any executable command!

PACKAGE I

Lattice MS-DOS C Compiler

C-SPRITE

LMK



PLUS Lattice Text Management Utilities (TMU)

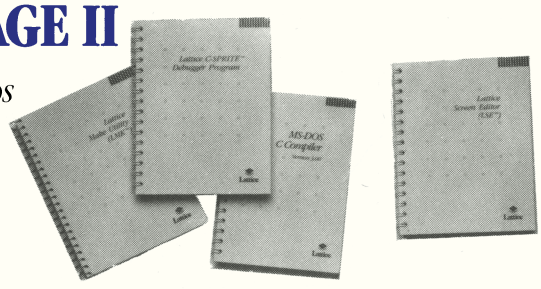
GREP, DIFF, EXTRACT, BUILD, WC, ED, SPLAT, and FILES—Eight software tools to help you manage your text files. Use TMU to search files for patterns, compare files and list their differences, create lists of file names, build batch files, display the number of characters and the checksum of files, and search and replace character strings. TMU includes a line editor, and TMU lets you list, copy, erase or remove files or entire directory structures meeting specified conditions. Because TMU is not language oriented, you can use it on files created by text editors or word processors. Files in C, Assembler, COBOL, Fortran, Pascal, or plain English can all be handled by TMU.

PACKAGE II

Lattice MS-DOS C Compiler

C-SPRITE

LMK



PLUS Lattice Screen Editor (LSE)

LSE is fast, flexible and easy to learn. Its multi-window environment provides all the editor functions you need including block moves, pattern searches, and cut-and-paste. LSE also offers special features for programmers such as an error tracking mode and three Assembly Language input modes. LSE lets you create macros and customize keystrokes, menus, and prompts to match your style and preferences.

Call (312) 858-7950 or return this order form today!

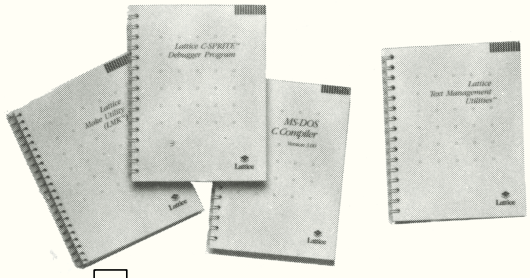
☐ Package I Only $699

Lattice MS-DOS C Compiler

C-SPRITE

LMK

TMU



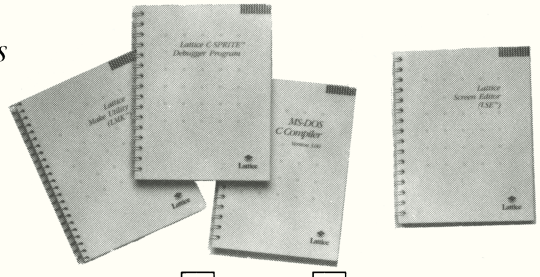
☐ Package II Only $699

Lattice MS-DOS C Compiler

C-SPRITE

LMK

LSE



☐ Payment Enclosed ☐ PO #_____ ☐ Bill My ☐ American Express ☐ MasterCard ☐ Visa

Name_____

Company_____

Address_____

City_____ State_____ Zip_____

Telephone_____

Expiration Date_____

Card Number_____

Signature_____

We'll save your sanity at a special Discount Rate!

InfoPro Systems™

3108 Route 10, P.O. Box 849, Denville, NJ 07834

Bulk Rate
U.S. POSTAGE
PAID
Cincinnati, OH
Permit No. 4742

Is C Driving You Buggy?

LATTICE, INC.
22 W 600 BUTTERFIELD RD.
GLEN ELLYN, IL. 60137

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 109 EAST HANOVER, NJ

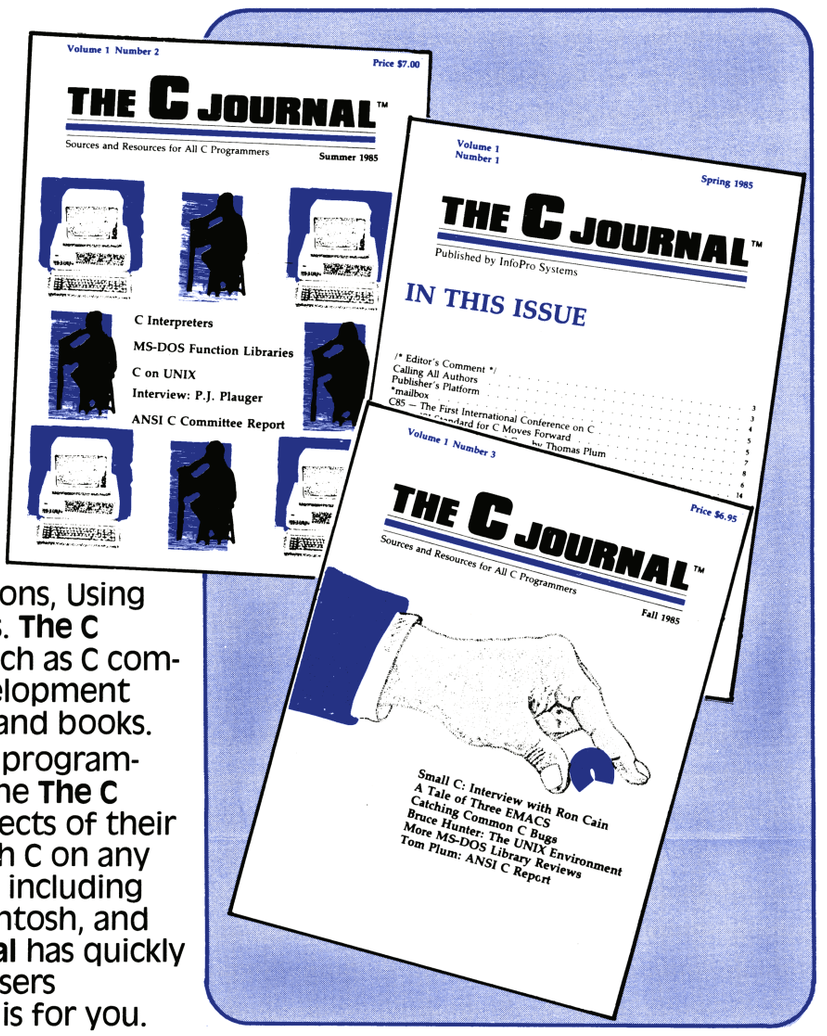
Postage Will Be Paid By Addressee

InfoPro Systems™
P.O. Box 849
Denville, NJ 07834

InfoPro Systems™

The C Journal is a new and exciting magazine devoted ***strictly*** to the C programming language. This high-quality publication is dedicated to programmers and software managers like yourself who use C. Every issue of **The C Journal** contains technical tutorials, programming tips, coverage of the ANSI C standards committee, C for UNIX users, interviews with the giants of the C industry, rumors and news. Feature articles have covered topics such as Common C Programming Errors, Pointers to Functions, Using ***setjmp*** and ***longjmp,*** and C Interpreters. **The C Journal** includes reviews of products such as C compilers, text editors, software tools, development systems, applications libraries, training and books.

Hobbyists, novice and experienced programmers, and software managers turn to the **The C Journal** for valuable assistance in all aspects of their work. **The C Journal** helps you work with C on any computer under any operating system, including MS-DOS, PC-DOS, UNIX, Xenix, CP/M, Macintosh, and Amiga. It's no surprise that **The C Journal** has quickly become ***the*** information source for C users everywhere. If you use C, **The C Journal** is for you.



BIG DISCOUNT & FREE BONUS TOO!

A full year of **The C Journal** normally costs $28, but if you **ACT NOW** you can get it at our Special Discount Rate of only $18 (that's over 35% off)! And as a special **FREE BONUS,** we will also send you a 3-color **C Language Reference Card** upon receipt of your paid subscription. This permanent, 8½" by 11" guide to the entire C language regularly sells for $5.95, but can be yours **FREE.**

This is a ***limited time*** offer, so complete the order form below and start your subscription to **The C Journal** TODAY!

Catch This Special!

Don't Wait Until C Drives You Buggy!

Insure your sanity today. Get a full year of **The C Journal** NOW (four big issues) while this special offer lasts, get your FREE C Language Reference Card, save $10, and tap into the best C information source on the market!

Yes! Save my sanity. Start my subscription today!

Name_____

Position _____

Company_____

Address _____

City _____ State_____ Zip_____

Country (if not US)_____

Phone ( )_____

Please add $9 for airmail delivery outside North America, or $6 extra for Canada or Central/South America.

For faster service on credit card orders, call **(201) 989-0570**

☐ My check for $18 is enclosed

CHARGE MY:

☐ MasterCard ☐ VISA

Card No.

Exp.

Signature

☐ BILL MY FIRM
I understand a $5 billing charge will be added.

Purchase Order No.

MONEY-BACK GURANTEE: Your satisfaction is assured or we will send you a refund for all unmailed issues.

Lattice, Incorporated
Post Office Box 3072
Glen Ellyn, Illinois 60138
312/858-7950
TWX 910-291-2190
TELEX 532253

FIRST-CLASS MAIL
U.S. POSTAGE
PAID
Glen Ellyn, IL
Permit No. 597

Current Revisions of Lattice Products

This list is included in Lattice Works to notify you of our current product revisions. Please be sure to register all your purchases with us so that you can be notified when new versions are released. Contact Lattice if you wish to receive the current revision.

Current Revisions:

Lattice MS-DOS C Compiler 3.00
PLINK86 Linker 1.47
Unicalc Electronic Spreadsheet 4.00
LMK "Make" Utility 2.20
C-FOOD Smorgasbord Library 2.13
Curses Screen Package 1.02
dBC II ISAM Library 1.07
dBC III ISAM Library. 1.07
CVUE Full Screen Editor 1.02
Panel 6.01
TMU Text Utilities 2.00
Lattice MS-DOS to 68000 C Cross Compiler 3.03
Lattice MS-DOS to Z80 C Cross Compiler 3.02
Lattice VAX/VMS to iAPX86 C Cross Compiler 1.27
Lattice SUN/UNIX to iAPX86 C Cross Compiler 1.27
Lattice VAX/UNIX to iAPX86 C Cross Compiler 1.27
VAX PLINK86 . . . . . 1.44
SUN PLINK86 . . . . . 1.44
CXREF 1.03
LSE 1.00
TopView Toolbasket 1.01
db-VISTA. 1.30
Pre-C. 1.34
FirsTime 1.04
C-SPRITE Source Debugger. 2.02
Instant-C 1.41
SecretDisk. 1.00
SideTalk 1.00

Amiga Products:

Lattice AmigaDOS C Compiler 3.03
Lattice MacLibrary 1.00
Lattice Text Utilities 1.00
Lattice Make Utility. 1.00
Lattice Screen Editor. 1.02
Lattice dBC III Library 1.00
Panel. 6.20
Unicalc Electronic Spreadsheet 1.00
C Cross Reference Generator 1.00
Lattice MS-DOS to AmigaDOS C Cross Compiler 3.03

Products No Longer Available:

Lattice C Compiler (CPM/86)
Lattice Windows
TMU Source

Lattice Works is published by Lattice, Incorporated and is edited by Dan Knopoff and Steve Hersee of Lattice, Incorporated.

Lattice Works will be sent to all of our registered customers. Please be sure to complete and return the registration card enclosed with your purchase.

Lattice® Works

Lattice

Spring 1986

Lattice Expands Sales Department

The more you need Lattice products, the more you need Lattice! We're responding to your needs by expanding Dorothy Gaden's Telemarketing Sales Department. We now have more phone lines, more space and more professionals to handle your sales order inquiries.

In addition to the larger telemarketing department, we have established a new Corporate Sales Department to increase our presence in OEM markets and large corporate computing operations. Under the direction of Wayne Nartker, the Corporate Sales Department will provide the level of service and volume discounts required by large-scale users of Lattice products.

Wayne has a special understanding of OEM and large corporate needs. Formerly Programming Manager at Lattice, Wayne has developed close working relationships with key OEMs. "In the past, large users and OEMs have come to Lattice through word of mouth," Wayne states. "Now we are in position to initiate the contacts.

"With our broad customer base, wide range of products and increased staff in all areas, we have a powerful story to present. The Corporate Sales Department will demonstrate how Lattice can be a valuable partner in large-scale computing efforts."

Wayne will also stress the broadening of relationships with current users. He will make personal visits to large users' sites—helping them use Lattice products, offering support and demonstrating how our new products can meet their needs.

"Lattice's maturity makes us an attractive source for hardware vendors and OEMs," Wayne pointed out. "We offer stability, support and a proven base of users."

Wayne emphasizes that by expanding commitments with OEMs, Lattice will provide software developers who use Lattice products with opportunities to transport their software to new hardware environments. "As a case in point, Commodore asked Lattice to develop a native C compiler prior to the release of the Amiga," Wayne relates. "Through our expanded sales efforts, you can be sure that Lattice C will be available on new machines in the future."

Our expanded sales department lets us give you prompt service when you call.

Great Scott!

Need a speedy answer to a question about a Lattice product? Get your version and serial number and call Scott Dickson, our new Supervisor of Customer Support.

Scott joined Lattice in September, bringing a combination of experience in sales, programming and support that translates to expert technical help for Lattice users. A former sales rep for Illinois Business Consultants (micro application aids), he has also worked as an independent programmer creating customized accounting applications.

With experience in customer contact, Scott knows how to communicate. And his programming background enables him to quickly grasp questions or problems, then clearly explain the answers.

He is committed to maximizing the effectiveness of the Customer Support group. "As the number of Lattice users increases, so will the size of the Customer Support group," Scott emphasizes. "It's important for our users to have a knowledgeable person on the other end of the line, and to get an answer quickly.

"We can usually supply an answer while the user is on the line. In most cases, we can solve the problem immediately by helping the caller find the answer in the manual. Sometimes we need a file containing test data for us to analyze, but even then, we answer as quickly as possible."

Scott Dickson, Supervisor, Customer Support

Scott says about 50% of the customer support calls concern the Lattice C Compiler, with the balance covering the other 40 products sold by Lattice. In cases where Scott must consult with the Lattice programming staff, Scott gets back to callers ASAP.

In between phone calls, Scott is busy answering questions via the mail or the BIX network. "C is a useful, powerful language, and there's a lot to it," Scott says. "But the constant challenges along with the opportunity to share problem-solving with people throughout the industry, make this job a lot of fun."

Lattice At COMDEX Fall

If you were in Las Vegas for COMDEX Fall, you know Lattice was there! Via exhibits, dealer/distributor receptions, press receptions and custom literature bags, the Lattice name was prominent at the computer industry's largest trade show.

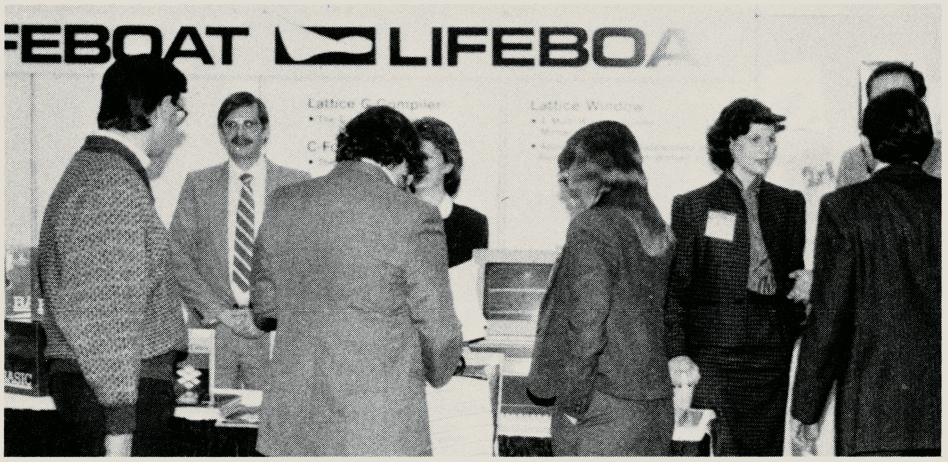
It was great to meet many of you—end-users, dealers, corporate and data processing department heads, hardware manufacturers, distributors and trade press editors—face to face. Our show contacts have resulted in expanded distribution of Lattice products, as well as greater name recognition. And it was at COMDEX that we met the people that enabled us to set up the BIX network support service.

Exhibiting in the Lifeboat Associates and Softsel booths, Lattice introduced 13 new products at the Show. The tremendous interest in our new products for the Amiga gratified our entire staff.

Sandwiched between dealer/distributor receptions was the "Lattice Press Express", a novel approach to capturing the eye of the media. At the close of the Show's second day, editors were invited to the Lattice suite, and we express-mailed press kits and materials back to their offices. The one-on-one meetings with key editors resulted in commitments for coverage of news stories and product reviews.

Many thanks to all who offered their time and support to Lattice at the Show. We look forward to seeing you again at COMDEX Fall '86.

Activity was brisk all day around our two exhibit locations.



Steve Hersee and Steve Twombly, Publisher of Amiga World magazine, meet at the Press Express reception.

Receptions after show hours meant constant activity during the evening hours, too. But we wouldn't have wanted it any other way.

Lattice Support Goes Online

The more you use Lattice products, the more you need Lattice! We're responding to your needs for increased support by offering a 24-hour on-line consultation service.

Through a special arrangement with McGraw-Hill's BIX (Byte Information Exchange) network, you can arrange a public or private answer concerning a Lattice product question.

Chances are, the answer to your question already exists on BIX. BIX averages nearly 200 calls per day on Lattice products, and we check the service regularly to assure you of a prompt answer.

You can "BIX-mail" your questions directly to Lattice using the Lattice logname ("jriley") for a private question, or you can post your question in the public area for Lattice or other users to answer.

With the BIX network, you get the advantage of Lattice support and more! You also get the experience of Lattice's many other users.

All you need is an interest in Lattice products, a modem, and Visa or MasterCard. The cost is only $25 for Byte magazine subscribers ($39 for non BYTE subscribers) plus $11 to $14 per hour for connect time.

To make your connection to BIX even easier, use our new Lattice communications package, SideTalk! Call us for details.

The BIX Connection

1. Call Tymnet at (800) 336-0149 to obtain the local access number
2. Call the local Tymnet access with your modem and type:

a	(a single lower-case "a") to tell the baud rate
byteneti	entered at the login prompt
mgh	entered at the password prompt
bix	to tell tymnet you want BIX
new	entered at the BIX prompt for name

3. BIX will ask you a number of questions. Once you are on, you will be able to get the latest information about any Lattice product.

If you have questions or problems about BIX, you can call their support line at 1-800-227-2983. If you have questions about our use of BIX, call Lattice.

Coming Attractions LOGISTIX

Lattice introduces Logistix—our new integrated software package that combines time management, database, spreadsheet, and graphics into one advanced business package.

Using Lattice Unicalc® as its base, Logistix expands the spreadsheet into a complete business planning and decision support tool. With Logistix, you can store, analyze, manipulate and present time elements in addition to numerical or statistical information.

The Logistix worksheet measures 2048 rows by 1024 columns and features intelligent memory management that allows maximum dimensions to be reached. You also get color support formatting options, more than 150 screens of context-sensitive help, full use of special function keys, automatic support of an 8087 coprocessor, and the ability to create macros. Data, formats, formulas, references and functions can be consolidated from Lotus 1-2-3, SuperCalc, and other Logistix files, or you can import dBASE, DIF, CSV and text files.

The Logistix **time management** functions give you the ability to perform critical path analysis for project scheduling with time units ranging from half-hours to years. You simply enter commands directly into the worksheet cells to define job records and schedule activities and resources.

Logistix' **spreadsheet** functions include 74 mathematical, logical, statistical, financial and calendar functions. Financial figures may be formatted with any of the major international currency symbols, and dates may be displayed in any of 14 different national formats. "What-If" tables with multiple variables give Logistix powerful modeling capabilities.

The Logistix **database** will store 2047 records by 64 fields. Record sorting, extraction, insertion, and deletion are supported along with full logical and mathematical operations which can be used in selections.

In addition, Logistix offers pie, bar, linked, 3D and other **graphic presentations** of worksheet data from graph commands entered directly into the worksheet cells. Logistix provides full support for a complete range of plotters and dot matrix printers. It even provides a sideways printout of the worksheet.

Logistix was created for us by Grafox, of Oxford, England. We are offering it to U.S. users for $450. It is available for the IBM PC, XT, AT, and compatible systems with 512K memory.

Call us today for more complete information on Logistix' features and to order!

Lattice Works (For C Standards)

As you know, Lattice has been an active player in the drive to develop an ANSI standard for the C language. Steve Hersee represents Lattice on the ANSI X3J11 C Standards Committee, and has reported its progress via his articles in UNIX/World magazine.

Lattice is proud to expand its efforts to the international front. Steve was elected as Convener (Chairman) of the International Standards Organization (ISO) Working Group at the ISO's meeting in Paris this past November. At the historic gathering, Steve represented the U.S. delegation for the C language.

Lattice, by its active participation in the standardization arena will help to insure that one useful standard will be adopted both in the USA and worldwide. While closer to home, Lattice has incorporated parts of the proposed standard in Version 3.0 of the Lattice C Compiler as part of our effort to continually improve the Lattice C Compiler. Users of previous versions of Lattice C will find it easy to benefit from the new C features.

Call us today for complete details on all the new features and advantages of the Lattice C Compiler Version 3.0.

Lattice Takes Legal Action

Lattice, Inc. announces that it has filed a lawsuit in the U.S. District Court, Albuquerque, New Mexico against Joe and Carole Petranovich, residing in Albuquerque.

The suit alleges that the defendants have infringed Lattice's copyright on its C Compiler software by illegally copying the program and offering copies for sale. The suit seeks appropriate damages as well as an injunction against future infringement.

PANEL FOR LATTICE C

1986 ROUNDHILL
Version 6.20D
Serial AM10-41279
AMIGA 880 FORMAT

Lattice

Lattice, Incorporated
© 1985 Lattice, Incorporated

This was brought to you

from the archives of

http://retro-commodore.eu